NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EVALUATING THE EFFECTIVENESS OF WATERSIDE SECURITY ALTERNATIVES FOR FORCE PROTECTION OF NAVY SHIPS AND INSTALLATIONS USING X3D GRAPHICS AND AGENT-BASED SIMULATION

by

Patrick Joseph Sullivan

September 2006

Thesis Advisor:	Don Brutzman
Thesis Co-advisor:	Curtis L. Blais

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2006	Master's Thesis

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Evaluating the Effectiveness of Waterside Security Alternatives for Force Protection of Navy Ships and Installations Using X3D Graphics and Agent-Based Simulation.	
6. AUTHOR Patrick Joseph Sullivan	
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAMES AND ADDRESSES Naval Facilities Engineering Service Center, Port Hueneme California (NFESC) Navy Modeling and Simulation Office, Washington, DC (NMSO)	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	A

13. ABSTRACT

The individuals charged with the task of planning, developing and implementing force protection measures both at the unit and installation level must consider numerous factors in formulating the best defensive posture. Currently, force protection professionals utilize multiple sources of information regarding capabilities of systems that are available, and combine that knowledge with the requirements of their installation to create an overall plan. A crucial element missing from this process is the ability to determine, prior to system procurement, the most effective combination of systems and employment for a wide range of possible terrorist attack scenarios.

This thesis is inspired by the work done by James Harney, LT, USN (2003). The thesis will expand the Anti-Terrorism Force Protection Tool developed during the original thesis by including the capability of testing force protection measures in multiple scenarios by utilizing models of force protection equipment and forces, virtual worlds of existing naval facilities, and terrorist agents that exhibit intent and behavioral characteristics which can test the effectiveness of the force protection equipment used.

The result of this work is a scalable and repeatable methodology for generating large-scale, agent-based simulations for AT/FP problem domains providing 3D visualization, report generation, and statistical analysis.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Virtual Environments, X3D, Discrete Event Simulation, Simkit, Force Protection, Anti-Terrorism, Extensible Markup Language, XML, Java, Extensible Modeling and Simulation Framework (XMSF), SAVAGE, Distributed Interactive Simulation DIS-Java-VRML, DIS-XML	207
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EVALUATING THE EFFECTIVENESS OF WATERSIDE SECURITY ALTERNATIVES FOR FORCE PROTECTION OF NAVY SHIPS AND INSTALLATIONS USING X3D GRAPHICS AND AGENT-BASED SIMULATION

Patrick Joseph Sullivan
Lieutenant, United States Navy
B.A., University of New Hampshire, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL
September 2006

Author:	Patrick Joseph Sullivan
Approved by:	Don P. Brutzman Thesis Advisor
Approved by:	Curtis L. Blais Thesis Co-advisor
Approved by:	Rudy Darken Chair, MOVES Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The individuals charged with the task of planning, developing and implementing force protection measures both at the unit and installation level must consider numerous factors in formulating the best defensive posture. Currently, force protection professionals utilize multiple sources of information regarding capabilities of systems that are available, and combine that knowledge with the requirements of their installation to create an overall plan. A crucial element missing from this process is the ability to determine, prior to system procurement, the most effective combination of systems and employment for a wide range of possible terrorist attack scenarios.

This thesis is inspired by the work done by James Harney, LT, USN: "Analyzing Anti-Terrorist Tactical Effectiveness of Picket Boats for Force Protection of Navy Ships Using X3D Graphics and Agent-Based Simulation" (Harney 2003). The thesis will expand the Anti-Terrorism Force Protection Tool developed during the original thesis by including the capability of testing force protection measures in multiple scenarios by utilizing models of force protection equipment and forces, virtual worlds of existing naval facilities, and terrorist agents that exhibit intent and behavioral characteristics which can test the effectiveness of the force protection equipment used.

The result of this work is a scalable and repeatable methodology for generating large-scale, agent-based simulations for AT/FP problem domains providing 3D visualization, report generation, and statistical analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACRONYMS AND ABBREVIATIONS

A*	A Star
AI	Artificial Intelligence
AMM	Avoidance Mover Manager
AT/FP	Anti-Terrorism / Force Protection
API	Application Programming Interface
ATO	Air Tasking Order
C2	Command and Control
CNI	Chief of Naval Installations
CRS	Congressional Reporting Service
DIS	Distributed Interactive Simulation
DES	Discrete Event Simulation
DOD	Department of Defense
DOE	Design of Experiments
ELV	Explosive Laden Vessel
ESPDU	Entity State Protocol Data Unit
FPO	Force Protection Officer
FSA	Finite State Automata
GUI	Graphical User Interface
GWOT	Global War on Terrorism
HTML	Hypertext Markup Language
IMM	Intercept Mover Manager
IEEE	Institute of Electrical and Electronics Engineers

JAXB	Java Architecture for XML Binding
JDOM	Java Document Object Model
JPEG	Joint Photographic Experts Group
LSVE	Large-scale Virtual Environment
M&S	Modeling and Simulation
MAS	Multi-Agent Systems
MOE	Measure of Effectiveness
MOP	Measure of Performance
MPC	Military Patrol Craft
NPS	Naval Postgraduate School
PCL	Property Change Listener
PNG	Portable Network Graphics
RHIB	Rigid Hull Inflatable Boat
SAVAGE	Scenario Authoring and Visualization for Advanced Graphical Environments
SMAL	Savage Modeling Analysis Language
SME	Subject Matter Expert
SSDF	Ship Self Defense Force
TCP	Terrorist Cell Planner
VRML	Virtual Reality Markup Language
W3C	World Wide Web Consortium
X3D	Extensible 3D Graphics Standard
XML	Extensible Markup Language
XMSF	Extensible Modeling and Simulation Framework

XSLT	Extensible Stylesheet Language for Transformations
ZMM	Zone Mover Manager

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge the financial support provided by the Naval Facilities Engineering Service Center (NFESC) and the Navy Modeling and Simulation Office (NMSO). Additionally, the author would like to thank the following people whose direct support made this thesis possible:

- My wife Vicki and daughters Sarah and Kaitlyn, for their patience, understanding, and support during this project.
- Don Brutzman and Curt Blais for their guidance and support during this thesis research.
- Jeff Weekley and Terry Norbraten for their willingness to listen and offer advice on numerous aspects of this thesis research.
- Rick Goldberg, Mike Bailey, Arnold Buss, and Alan Hudson for providing the tools needed to complete this research and for their assistance in implementing new ideas and concepts.
- Planet 9 studios for creating the virtual worlds for this project.
- John Haussermann, Thomas Otani, Paul and Susan Sanchez, Chris and Rudy Darken, Don McGregor, Roberto Szechtman, Moshe Kress, and John Hiles for their excellence in teaching and for providing me with the required knowledge to perform this research.
- Charles Lakey, Jason Jones, Dimitrios Myttas, Rommel Toledo, Travis Rauch, Adrian Armold, Doug Wahl, Sean Lewis, Tom Scola and Bryan Lee for making this experience more enjoyable through their company, conversations, and laughs.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

Individuals charged with the task of planning, developing and implementing force protection measures, both at the unit and installation level, must consider numerous factors in formulating the best defensive posture. Currently, force protection professionals utilize multiple sources of information regarding the capabilities of systems that are available, and combine that knowledge with the requirements of their installation to create an overall plan. A crucial element missing from this process is the ability to determine, prior to system procurement, the most effective combination of systems and employment for a wide range of possible terrorist attack scenarios.

This thesis expands the Anti-Terrorism / Force Protection (AT/FP) Tool developed by Lieutenant James Harney (Harney 2003) by providing the ability to evaluate security alternatives utilizing models of force protection assets, existing naval facilities, and terrorist agents. The tools and methodologies developed in this thesis are designed to address the potential needs of three end users: Naval Installation Security Planners, Harbor Operations Support Staff and shipboard Force Protection Officers (FPOs). This work can provide each of these users with a means to test the effectiveness of force protection equipment and assets in a simulated environment. The result of this work is a scalable and repeatable methodology for generating large-scale, agent-based simulations for AT/FP problem domains providing 3D visualization, report generation, and statistical analysis.

B. OVERVIEW

The protection of United States Navy assets and installations against terrorist attacks has been a persistent focus of the United States government since the USS Cole attack in Aden Harbor, Yemen on October 12, 2000 (CRS 2001). The Cole attack was a primary motivation for Harney's work. On August 19, 2005, four years after the Cole attack, another terrorist attack was attempted on a U.S. Navy ship in port. Three rockets

were fired at the USS Ashland, an amphibious assault ship conducting a port visit in Aqaba Jordan, and narrowly missed the bow.

Both of these attacks occurred while the ship was in port. Harbors present a very vulnerable environment for military ships. Potential adversaries can utilize a variety of methods to conduct an attack while pier-side ships are limited in their ability to perform self-defense. The National Maritime Strategy, released in September 2005, highlighted the complexity of this problem stating that terrorist organizations have demonstrated the ability to:

> …develop effective attack capabilities relatively quickly using a variety of platforms, including explosives-laden suicide boats and light aircraft; merchant and cruise ships as kinetic weapons to ram another vessel, warship, port facility, or offshore platform; commercial vessels as launch platforms for missile attacks; underwater swimmers to infiltrate ports; and unmanned underwater explosive delivery vehicles. (White House 2005)

As the Global War on Terrorism (GWOT) continues, the threat to naval assets and harbor facilities is likely to be a major focus for the organizations charged with providing their protection.

C. MOTIVATION

One of the challenges of AT/FP planning is that while the objectives of the adversary may be imaginable, the ability to foresee how these objectives will be effectively opposed is not. Additionally, due to finite resources, it is impossible to defend against all terrorist attack scenarios. This situation requires force protection professionals to accept certain levels of risk and to evaluate the most effective defenses based on what are perceived to be the most vulnerable assets and potential attacker's most likely courses of action.

The Chief of Naval Installations (CNI) identified balancing this risk with the cost of preventing likely attacks as a critical component of future installation planning. In his twenty-five year plan for Naval Installations entitled Navy Ashore Vision 2030, the CNI acknowledged the need to:

"Develop and sustain a security capability that balances risk with cost to reasonably ensure protection of mission, mission support requirements, and large personnel concentration facilities from terrorist threats." (CNI 2005)

Modeling and Simulation (M&S) technology is an underutilized technology in this problem area, despite being well suited for evaluating these cost and. risk decisions. M&S technologies can be used to create 3D visualizations of complex scenarios while also providing statistical analysis for a variety of AT/FP scenarios. This work applies state-of-the-art and emerging M&S capabilities to provide better analytic support tools for this critical area.

D. OBJECTIVES

This thesis leverages web-based modeling, multiple technologies, and diverse complimentary projects developed at the Naval Postgraduate School (NPS) to explore their application to AT/FP scenario generation and analysis. The objective of this research is to create a repeatable methodology for generating AT/FP scenarios through the creation of software tools and exemplar models that provide a means to conduct harbor security studies, visualize the scenario outcome, and gain quantitative insight into problems of interest using statistical analysis.

E. THESIS ORGANIZATION

Chapter II reviews background technologies and related work used during this research effort. For each item referenced, a short description is provided to give the reader a baseline understanding of topics that are referenced throughout the remainder of the thesis. Chapter III outlines both the problem and research approach that are used to accomplish this work. Chapter IV discusses agent-based simulation within the context of AT/FP scenario authoring. Chapter V outlines how the ideologies presented in earlier chapters were created using new technologies recently developed at NPS. Chapter VI discusses the approach used for creating 3D visualizations of the simulation. It also provides examples of how 3D visualization was used for developing abstract simulation components for a discrete event model. Chapter VII introduces Savage Studio, a

graphical user interface (GUI) based scenario-authoring tool, designed to autogenerate both discrete-event models and web-based Extensible 3D Graphics (X3D) graphics of a complete AT/FP simulation. Chapter VIII provides an example end-to-end demonstration outlining how the various components of the preceding chapters can be combined to create a large-scale, real-world simulation. Chapter IX provides a summary of the conclusions from this work and provides recommendations for future work. The appendices provide examples and additional information as appropriate.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

This chapter provides a conceptual description of the technologies and related work that were leveraged to complete this thesis. The following sections are intended to provide the reader with a basic understanding of the multiple resources leveraged and are not intended to be all-inclusive explanations. References are provided for further investigation into each subject area.

B. MODEL AUTOGENERATION AND SCALABLE, REPEATABLE USE OF 3D GRAPHICS

The autogeneration of virtual environments for Department of Defense (DOD) applications is a recent concept. The technologies used for this application are beneficiaries of multiple bodies of work by Naval Postgraduate School Master's students. (Murray & Quigley 2000) established a methodology for creating 3D visualizations of air attack plans by translating a military Air Tasking Order (ATO). This approach was expanded to other domains such as autogeneration of a battle space from military message traffic (Nicklaus 2001), and 3D visualizations of tactical communication plans (Hunsberger 2001). The common thread in these research efforts was to automatically convert existing military data streams leveraging M&S technologies, thus providing war fighters with a greater understanding of their environment and enhance their decision making process.

In addition to the work cited above, (Brutzman 2003) outlines how "all manner of 3D objects can be modeled, animated and manipulated, in a scalable and repeatable fashion, in support of distributed large-scale virtual environments (LSVEs)." The knowledge gained from these works, and the ongoing construction of 3D models and software related projects, has made this thesis work possible.

C. DISCRETE EVENT SIMULATION

1. Methodology and Notation

Discrete Event Simulation (DES) is a modeling approach that is designed based on the events of a simulation. In DES, simulation time is advanced according to the next-event rule. A list of future events known as the Event List is maintained by the simulation. In this approach time is advanced based on the scheduled time of the next event instead of through small, constant duration time increments. In addition to events for time control, DES models have parameters and state variables. Parameters are elements that do not change and do not have the possibility of changing in the course of a single simulation replication. State variables stay constant during the time interval between events, and may change value during the instantaneous computation of the event according to a predefined state transition function. Events themselves are thus used to define state-transition functions (Buss 2004).

Event-graph methodology is an approach that allows for a structured, formalized representation of DES models (Schruben 1983). The use of event graphs allows a modeler to represent all aspects of a model in one graphical object. Figure 1 depicts an example event-graph model representation.

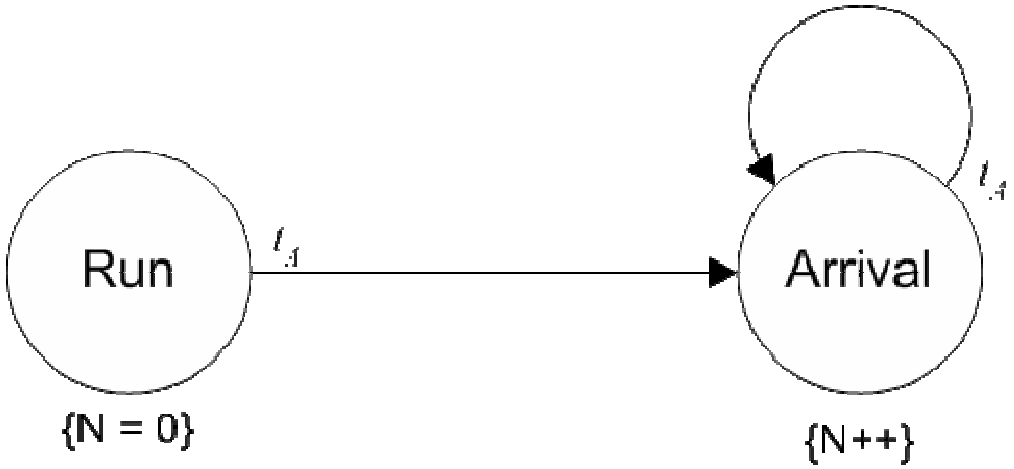


Figure 1. Depicts a simple example of event-graph notation. (from Buss 2004)

Circles are named events which instantaneously perform the state-variable computations shown in brackets. Arcs schedule events to occur after the labeled time delay.

As seen in Figure 1, events are annotated with circles and have identifying names (e.g., Run, Arrival). In this example the variable 'N' represents the total number of arrivals and is a state variable. The variable t_A represents the inter-arrival time and is a scheduling delay. The arcs between events are the final major component in an event graph model and represent the scheduling relationships between events.

Following this method, complex DES models can be created and represented as depicted in Figure 2. Arcs bisected by a squiggly line indicate a precondition prior to scheduling. Dotted arcs cancel previously scheduled, superseded events. Boxed values shown on top of an arc indicate passed parameters. It is worth noting that all Simkit event graphs have one additional special event, the Run event. This event is used by the simulation to reset the starting conditions for the model for each replication of the simulation.

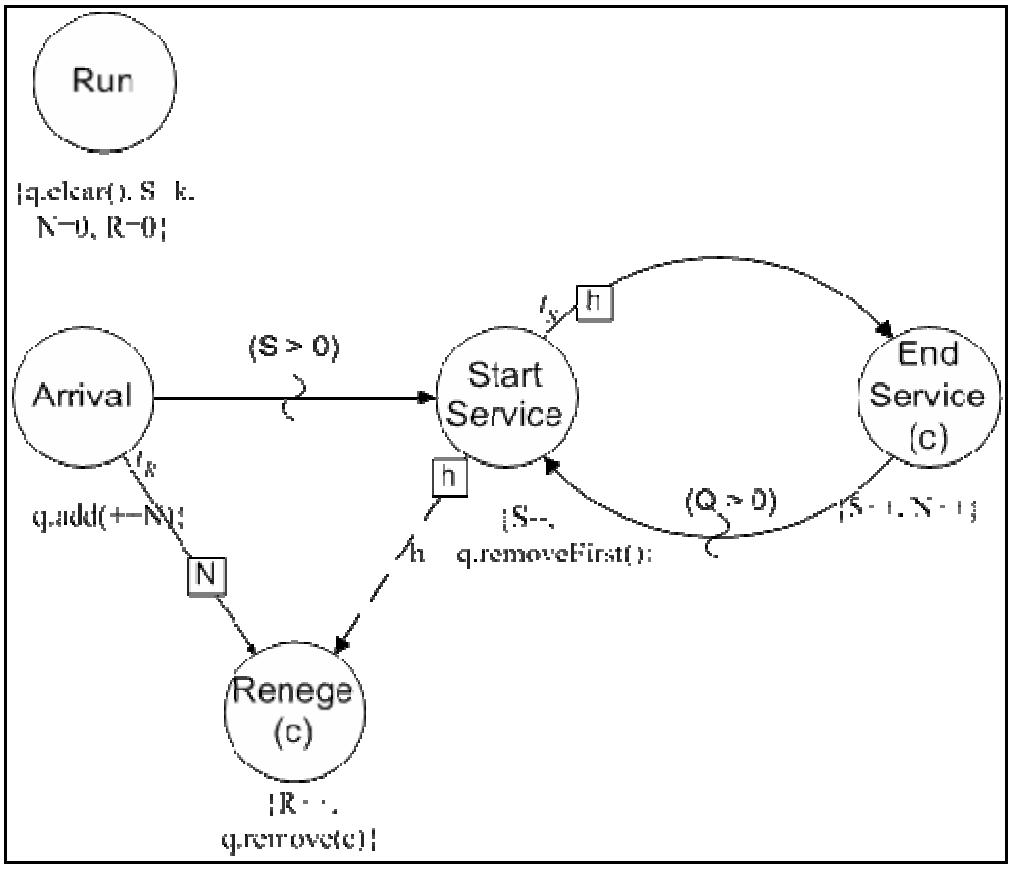


Figure 2. Depicts a more complex event graph model of a simple server process, Simple Server with Reneges. (from Buss 2004)

DES was selected for this project because of its potential to generate a large number of replications of complex scenarios, achieving large durations of simulation time in a short period of computational run time.

2. Simkit

Simkit is an open-source application programming interface (API) written in the Java programming language. Simkit was developed to implement computer simulations based on the event graph methodology previously discussed. This API allows the implementation of event graph based DES and provides a means to conduct statistical analysis of the simulation. Simkit also has the ability to support GUI programming to model and display 2D entities. Simkit's library of 2D mover and sensor objects, shown in Figure 3, provide the ability to create and visualize entity level simulations.

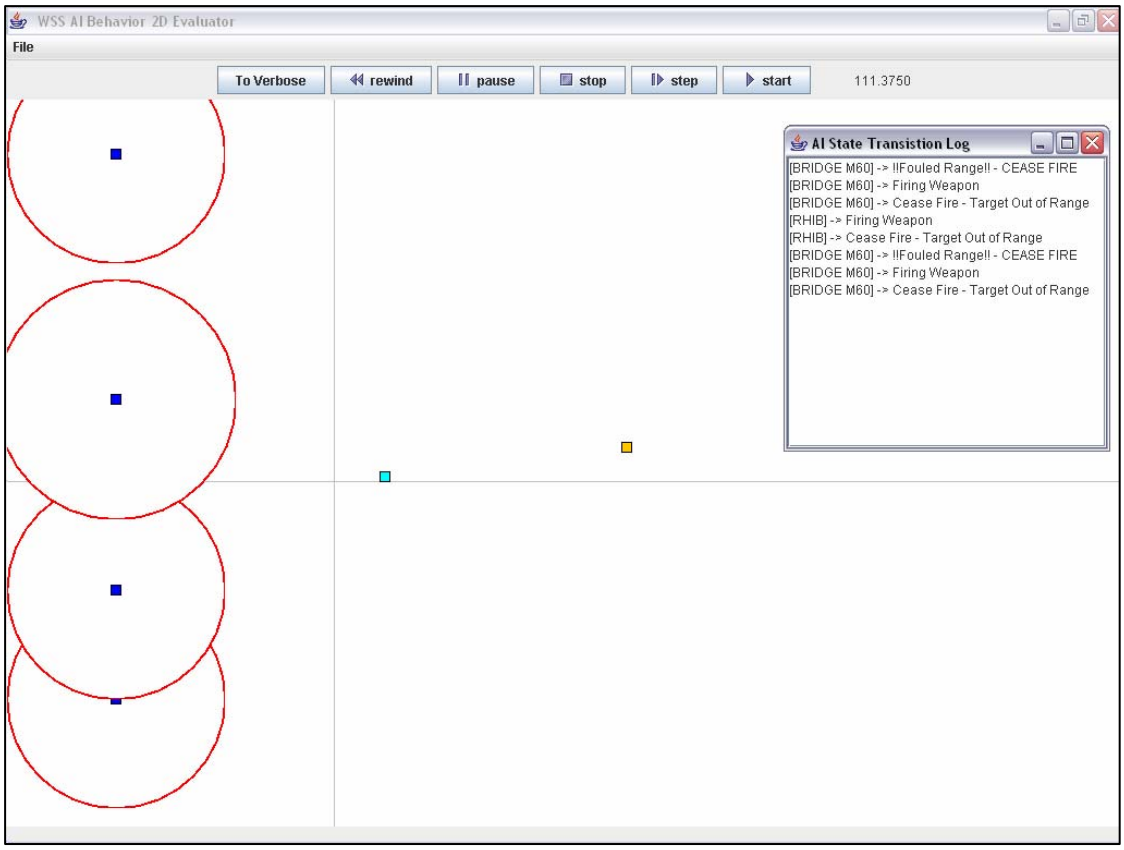


Figure 3. Example 2D display of simple mover and detection simulations using the Simkit API

Simkit is a stable, well-documented Java API that has been used in a variety of applications. The discrete event software discussed below is all built upon the foundation provided by Simkit. Information, source code, and examples of the Simkit API can be found at http://diana.gl.nps.navy.mil/Simkit/ (accessed August 2006). A complete list of the Master's thesis work that uses Simkit is provided in Appendix H and online at http://diana.gl.nps.navy.mil/~ahbuss/#STUDENTS (accessed September 2006).

3. Diskit

Diskit is a Java API that expands the Simkit mover and sensor libraries by including a third dimension. The API also exposes Simkit to the Distributed Interactive Simulation (DIS) protocol. (Singhal & Zyda 2000) provide an excellent discussion of the DIS protocol and implementation considerations related thereto. These two features enable computer simulations to animate 3D virtual environments greatly enhancing the visual representation experience. In the simple example shown in Figure 3, the navy blue boxes to the left represent a defender manning a weapon on a ship. The red circles

represent the firing zone for each defender. Figure 4 shows how the added functionality of the Diskit API enhances the visual representation.

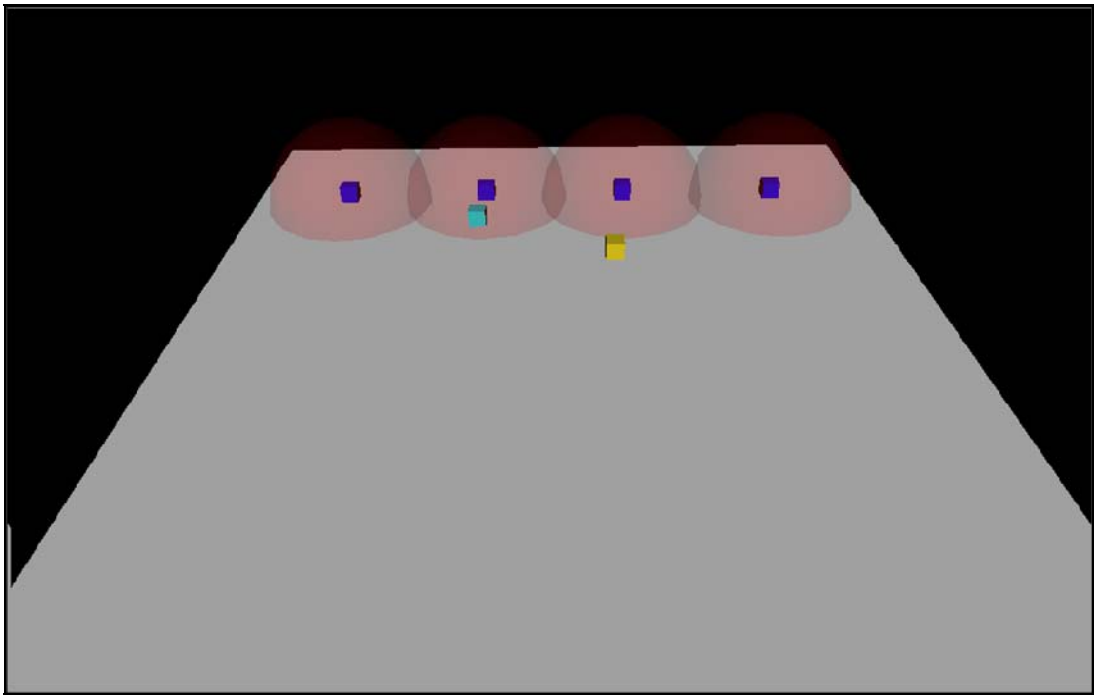


Figure 4. Simple 3D Visualization of Simkit mover and sensor objects enhanced by Diskit

4. Viskit

While Simkit and Diskit provide the ability to create computer simulations written in the Java programming language, they also require that the user be both proficient in Java programming and familiar with the API's. These two requirements limit the number of users that might benefit from the technology. Viskit is an open source graphical user interface (GUI) and visual programming methodology, aimed at reducing the knowledge required for using Simkit and Diskit. Viskit allows users who are familiar with the event graph methodology to create computer based simulations.

The Viskit GUI allows users to create event graphs of models that are representative of the hand-drawn examples shown in Figures 1 and 2. The event-graph editor shown in Figure 5 is familiar to users who understand the methodology and is far easier to implement than writing Java source code.

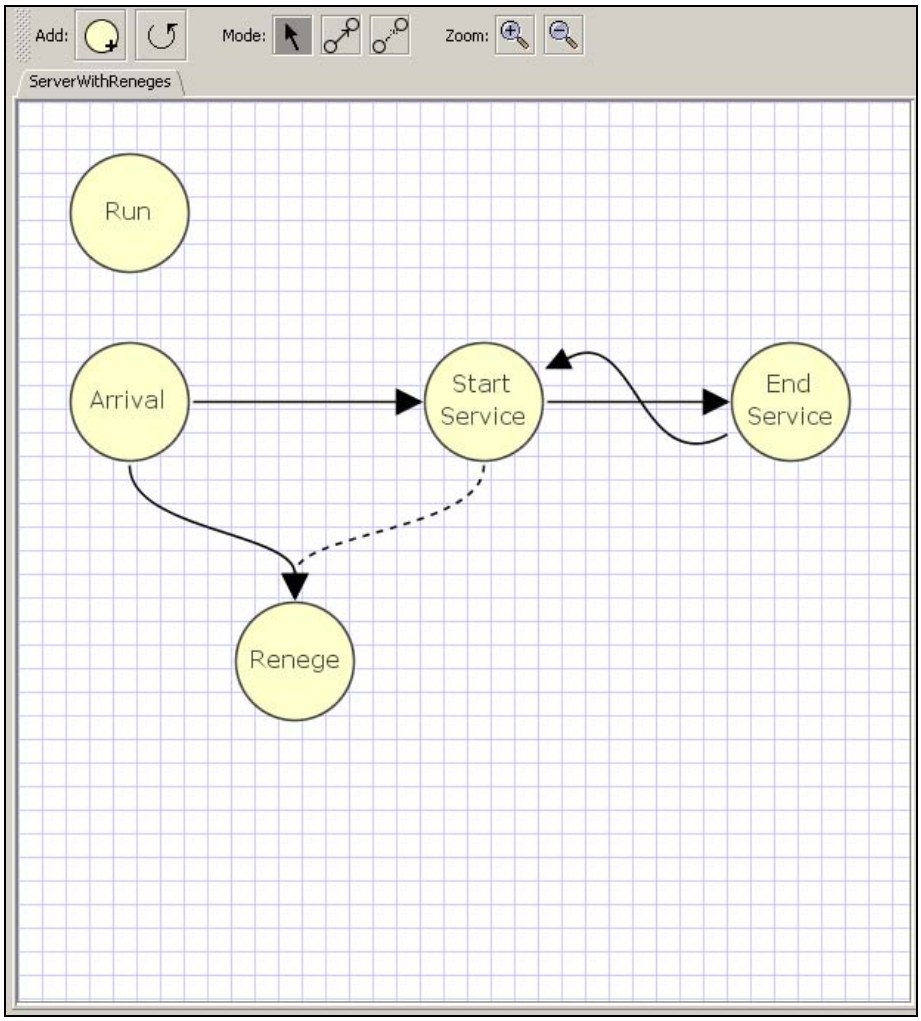


Figure 5. Simple Server with Reneges event graph displayed in the Viskit event graph authoring panel.

The Viskit interface also allows modelers to create more complex event graph representations without the need to worry about the complexity of the generated source code. The combination of Simkit, Diskit, and Viskit serve as the foundation for all simulation development in this project.

D. ROLE OF 3D VISUALIZATION IN DISCRETE EVENT SIMULATION

The utility of 3D visualization for DES models has been discussed and debated in various academic circles. Traditionally only 2D visualizations (such as the Simkit mover example in Figure 3) have been used for DES. As computer technology has evolved, 3D

visualization has emerged as a primary output for DES. A recent survey conducted by (Akpan & Brooks 2005) attempted to identify how useful 3D graphics were for DES models. This survey polled academics and organizations that use DES regarding the use of 2D or 3D visualization as an output for their simulations. The findings indicated that ultimately the true utility of the model was in the ability to derive numerical analysis from the results. The respondents indicated that the type of display used was immaterial if the model driving the visualization was not created properly. However, a majority of the survey respondents indicated that a 3D representation was invaluable for model verification and testing, and afforded observers a better understanding of the model environment. The survey concluded that as 3D graphics and computing resources continue to evolve, it is likely that more DES modelers will use 3D visualization.

E. X3D GRAPHICS

Extensible 3D graphics (X3D) is the computer graphics format used for this thesis. X3D is a royalty free, ISO-ratified format for quick and easy sharing of real-time 3D models, visual effects, behavioral modeling, and interaction. Using X3D, high-quality 2D, 3D and video information can be easily incorporated into technical publishing, maintenance manuals, websites, database applications, visual simulations, navigation systems and many other professional and consumer uses. (X3D 2006) A detailed explanation of X3D Graphics is provided in (Brutzman & Daly 2006). Chapter VI provides a detailed description of the X3D graphics considerations for this thesis.

F. THE JAVA PROGRAMMING LANGUAGE

1. Java Architecture for XML Binding (JAXB)

JAXB is an API that allows an XML schema to be bound to Java source code. JAXB enables the transformation of XML documents into strongly typed data structures accessible via executable java source code. In this project Viskit event-graph models are saved as XML documents. These XML documents are structured by and validated against a schema that is modeled based on the Simkit API. Utilizing JAXB Viskit classes take this XML representation and transform it into the Java source code which in turn utilizes the Simkit and Diskit libraries.

Figure 6 shows the XML tree display of Viskit. The event graph depicted is the Arrival Process model from Figure 1. This tree clearly displays the major components of an event graph model.

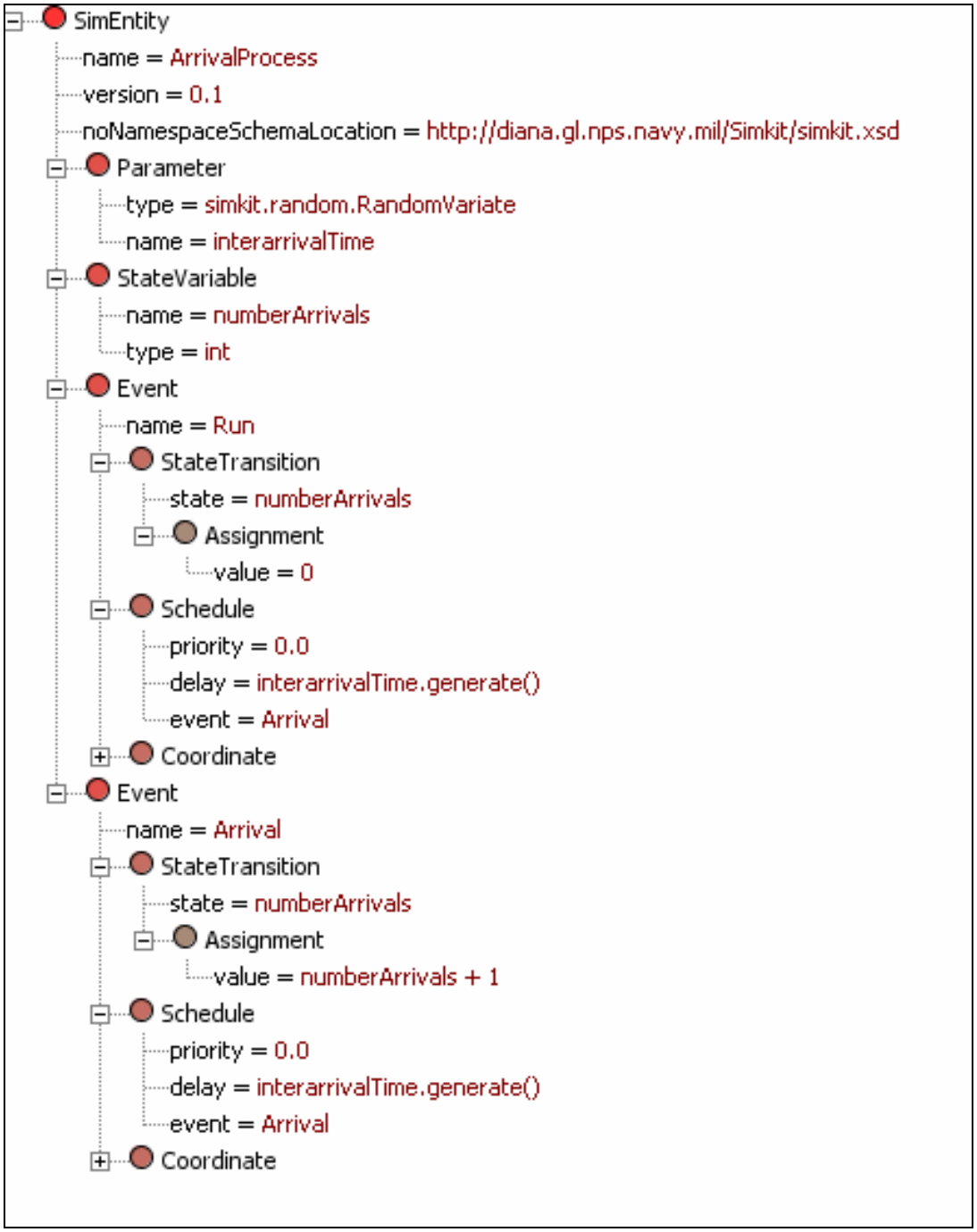


Figure 6. Depicts the Viskit event graph tree view corresponding to the Arrival Process event graph shown in Figure 1.

Figure 7 shows the corresponding Simkit-based Java source code that is autogenerated by Viskit for the Arrival Process event graph.

```
package examples;

import simkit.*;
import simkit.random.*;
import java.util.*;

public class ArrivalProcess extends SimEntityBase {

    private simkit.random.RandomVariate interarrivalTime;

    protected int numberArrivals;

    /** Creates a new instance of ArrivalProcess */

    public ArrivalProcess(simkit.random.RandomVariate interarrivalTime) {

        setInterarrivalTime(interarrivalTime);
    }

    /** Set initial values of all state variables */
    public void reset() {
        super.reset();

        /** StateTransitions for the Run Event */
        numberArrivals = 0;
    }

    public void doRun() {
        firePropertyChange("numberArrivals",numberArrivals);
        waitDelay("Arrival",interarrivalTime.generate(),new Object[]{},0.0);

    }

    public void doArrival() {
        /* StateTransition for numberArrivals */
        int _old_NumberArrivals = getNumberArrivals();
        numberArrivals = numberArrivals + 1;
        firePropertyChange("numberArrivals", _old_NumberArrivals,getNumberArrivals());
        waitDelay("Arrival",interarrivalTime.generate(),new Object[]{},0.0);
    }

    public void setInterarrivalTime(simkit.random.RandomVariate interarrivalTime) {
        this.interarrivalTime = interarrivalTime;
    }

    public simkit.random.RandomVariate getInterarrivalTime() {
        return interarrivalTime;
    }

    public int getNumberArrivals() {
        return  numberArrivals;
    }
}
```

Figure 7. Java source code for the Arrival Process event graph generated by Viskit using JAXB.

JAXB allows Viskit to translate XML structured data, which in this case is an event graph model, into executable Java source code. More information on JAXB can be found at http://java.sun.com/webservices/jaxb/ (accessed August 2006).

2. Java Document Object Model (JDOM)

JDOM is another open-source Java API that allows programmers to access, create, modify, and export XML documents using Java source code. JDOM is used for various purposes in this project. Appendix C provides an example use case of JDOM. This API bridges the gap between data sources that are in XML (e.g., Viskit, X3D Graphics) and the Java source code of Viskit, Diskit, and Simkit. Additional information on JDOM is available through the project website http://www.jdom.org (accessed August 2006).

3. JFreeChart

The JFreeChart API is an open-source API that allows Java programs to generate professional quality charts. Figure 8 shows an example chart generated using this API. In addition to generating a chart object, JFreeChart provides the capability to create and save picture files of the object in either Portable Network Graphics (PNG) or Joint Photographic Experts Group (JPEG) format. An example implementation using JFreeChart is provided in Appendix D. NPS purchased the documentation manual for this project and the quality of the documentation was excellent. Additional information on the JFreeChart project, including how to obtain supporting documentation is available at http://www.jfree.org/jfreechart/ (accessed August 2006).

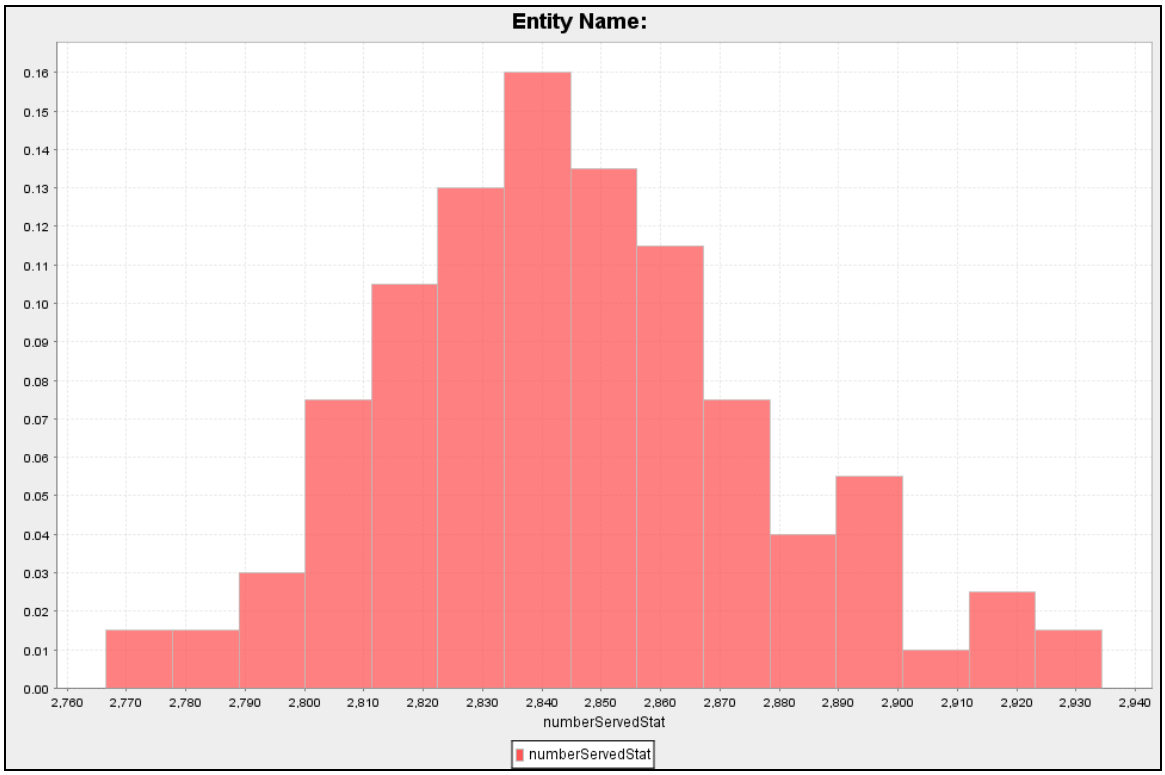


Figure 8. JFreeChart output example. Number of customers served histogram for Simple Server with Reneges event graph model.

G. DIS-JAVA-VRML

DIS-JAVA-VRML is a Java API that implements the DIS Protocol and allows Java applications to send DIS formatted information to X3D and VRML graphics models. Diskit and Viskit use this API to create and transmit information via the internet to the Xj3D browser displaying a virtual environment. DIS is an excellent choice for web based applications since it is an established Institute of Electrical and Electronics Engineers (IEEE) standard and allows users from different locations to collaboratively participate in the same simulated environment.

Detailed descriptions of DIS can be found in (Singhal & Zyda 2000) and (Hunsberger 2001). Specific information regarding the DIS-JAVA-VRML API can be found at http://web.nps.navy.mil/~brutzman/vrtp/dis-java-vrml/ (accessed August 2006).

H. XJ3D OPEN SOURCE PROJECT FOR X3D

Xj3D is an open source X3D browser and application that is written entirely in Java. In addition to a standalone version of the browser the API supports imbedding the browser in applications. This browser has been used for a number of applications at the Naval Postgraduate School and is the primary means to view X3D graphics for this project. A number of NPS project efforts helped to support the development of this codebase. Additional information on Xj3D can be found at http://www.xj3d.org (accessed August 2006).

I. VIZX3D / FLUX STUDIO SCENE AUTHORING TOOL

Flux Studio (formerly VizX3D) is a proprietary, feature rich X3D graphics authoring tool. The tool is a fraction of the cost of most tools of this type but provides exceptionally high levels of authoring capability and performance. This tool was used throughout this project to create and manipulate large-scale environments as well as experimentation with scenario development.

Figure 10 shows an example of the interface displayed in the new version of Vizx3D called Flux Studio.

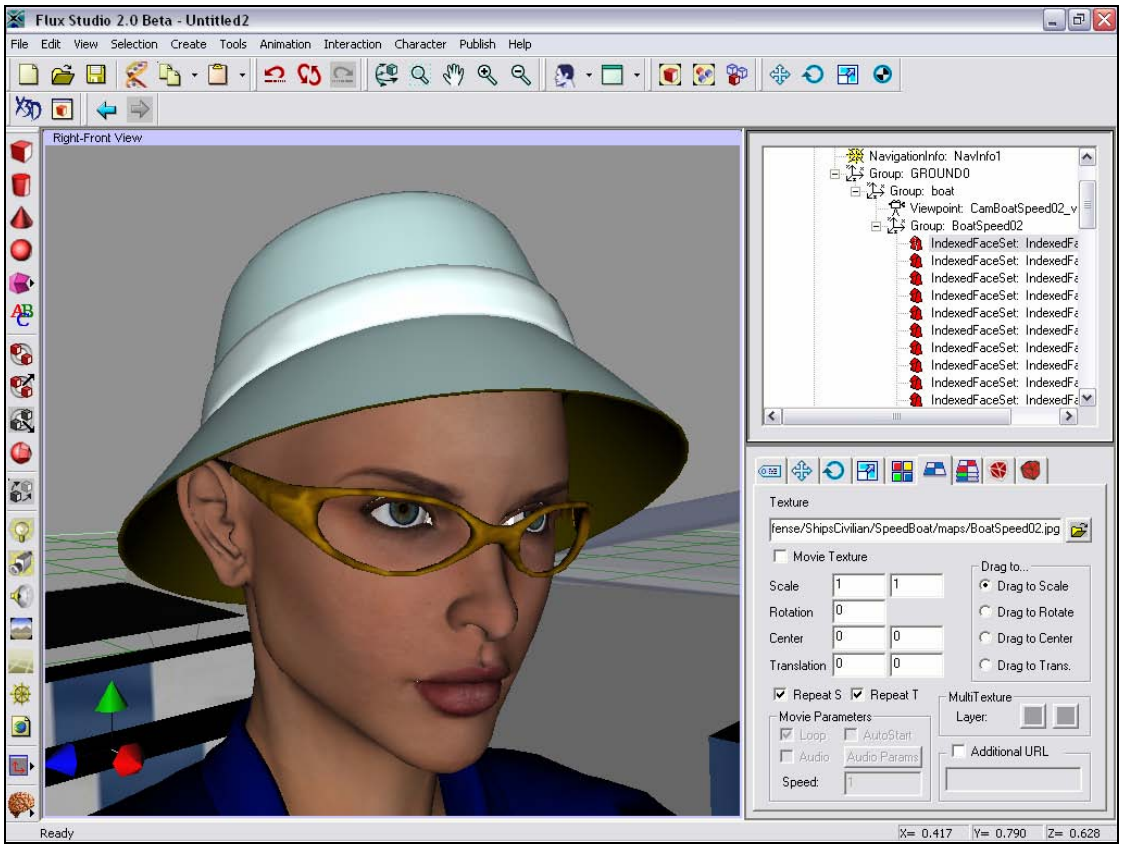


Figure 9. Flux Studio 2.0 (formerly VizX3D) screen capture showing a close up of a female terrorist avatar created for the AT/FP project.

In addition to a robust authoring capability that fully integrates the entire X3D scene graph, Vizx3D supports the import and export of multiple file formats. NPS project efforts helped to support the development and debugging of this tool. Additional information for this product and trial versions are available at http://www.mediamachines.com/ (accessed August 2006).

J. WINGS3D AUTHORING TOOL

Wings3D is an open-source 3D modeling tool that allows users to create complex geometric models with an intuitive, highly capable interface. It is important to note that Wings is designed to create large wire frame meshes and is not designed to optimize X3D content. Models created in this tool normally require further modification to optimize performance in an X3D scene graph editing tool.

Wings3D has a VRML export capability which allows users to create models and export them in a format that can then be used by other X3D modeling tools. Figure 10 shows the Wings3D interface. The Wings3D program and project information can be accessed at http://www.wings3d.com (accessed August 2006).

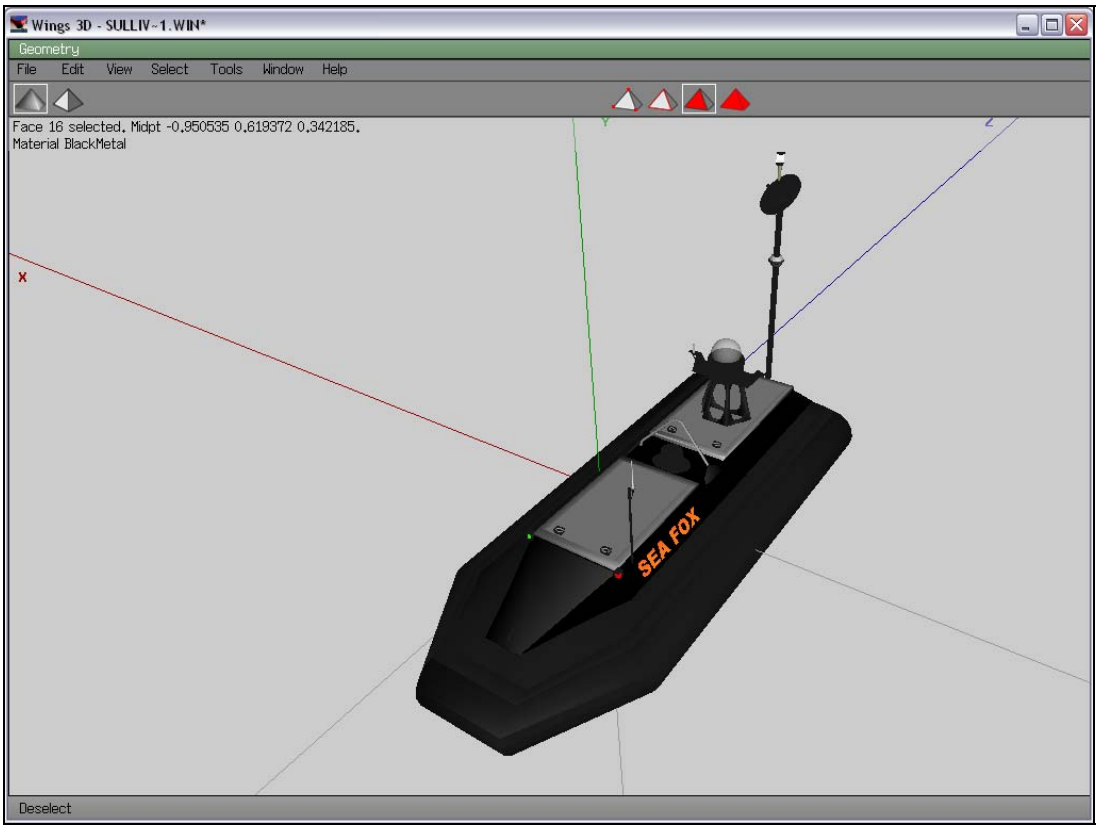


Figure 10. The Wings3D interface displaying an Unmanned Surface Vehicle (USV)

K. AGENT-BASED SIMULATION

Agent-based simulation was a logical choice for modeling the diversity of tactical entities needed for this thesis topic. The harbor environment involves a large number of autonomous entities performing their duties. Each entity has the ability to react to the environment and situations as they occur.

An agent-based approach affords analysts the opportunity to create complex scenarios with agents that have the ability to act independently and react to various conditions based on the current state of their surroundings.

Multi-Agent Systems (MAS) are defined by Ferber as "an electronic or computing model made up of artificial entities which communicate with each other and act in an environment" (Ferber 1999). Ferber formalizes this concept by identifying the key components of multi-agent simulations in a simple formula:

MAS = {Environment, Objects, Agents, Relations, Operations, Laws}

A detailed overview of multi-agent system components and usages is located in (Osborne 2002). Many potential implementation environments might be used to implement MAS. The NPS AT/FP project described in this thesis uses Viskit and Simkit to implement a large-scale networked MAS.

L. EXTENSIBLE MODELING AND SIMULATION FRAMEWORK (XMSF)

The Extensible Modeling and Simulation Framework (XMSF) is defined as a set of Web-based technologies, applied in an extensible framework, that enable a new generation of M&S applications to develop, emerge, and interoperate (Brutzman, Pullen & Zyda 2002). The primary subject areas for XMSF consist of: 1) Web/XML, 2) Internet/networking, and Modeling and Simulation (M&S). The XMSF architectural principles served as the fundamental basis for all technology design and integration in this thesis. The principal institutions leading the research and development of XMSF are: Naval Postgraduate School (NPS) Moves Institute, George Mason University (GMU) NetLab, Science Applications International Corporation (SAIC), and Old Dominion University (ODU) Virginia Modeling, Analysis & Simulation Center (VMASC) Battle Lab (Brutzman et al., 2002).

M. SAVAGE MODELING ANALYSIS LANGUAGE (SMAL)

Previous work on the autogeneration of 3D content has demonstrated the ability to create large virtual environments by leveraging existing archives of 3D model content. An example of such an archive is the Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE) 3D model archive. One of the greatest deterrents for using 3D graphics, as found in (Akpan & Brooks 2005), is the time and expertise required to create virtual environments.

The research in (Rauch 2006) addresses this problem by creating "an XML metadata standard to collect and organize the information necessary to create and populate a tactical 3D virtual environment: the Savage Modeling and Analysis Language (SMAL)." SMAL seeks to automate the 3D authoring process by embedding information about a model inside the model itself. Using this information tools such as Savage Studio, discussed in Chapter VIII, can autogenerate large 3D environments.

N. SUMMARY

This chapter has provided an overview of the technologies and prior work that were used for completion of this thesis. The reader is encouraged to explore references for additional background information.

THIS PAGE INTENTIONALLY LEFT BLANK

III. OVERVIEW OF THE PROBLEM

A. INTRODUCTION

Multiple technologies have been developed that have greatly enhanced the ability to create large-scale virtual environments (LSVEs) that can be used to enhance tactical-level decision making and planning. Unfortunately, few of these technologies are readily available or used on a day to day basis by military personnel. Incorporating M&S technologies into daily military operations is a missed opportunity often fueled by the complexity of the systems used.

A need exists to have GUIs that allow users with undergraduate collegiate level computer skills to interact with and create simulations. Intuitive interfaces would enable these individuals to not only plan and develop scenarios but also to verify, view, and interact with the simulation environment.

In (Harney 2003) the need to expand M&S technologies to create larger, repeatable, and scalable AT/FP scenarios was identified as recommended future work. This thesis aims to fully integrate many of these technologies to achieve that goal.

B. PROBLEM STATEMENT

The primary problem addressed by this work is to identify how to combine developed DES and M&S technologies to create virtual environments designed for AT/FP problems. Within the domain of AT/FP this work focuses specifically on the security of harbors, ships, and the waterfront area where ships operate. This research seeks to develop a repeatable and scalable methodology that can be used for harbor defense or any domain where DES, MAS, and a 3D graphical visualization is desired.

C. PROPOSED SOLUTION AND RESEARCH FOCUS

The proposed solution to this problem seeks to take full advantage of the multiple M&S technologies recently developed at NPS to create AT/FP scenarios. The first challenge for this work is to define a standard approach for using DES event graph models to model entity-level behaviors. The initial approach to this problem was to

identify the key behaviors to model and to map that behavior set to event graph models. These models were used to populate a behavior definition library to provide example implementations and a resource for future applications.

The second phase of this approach was to expand upon various 3D graphic model archives to provide the ability to create large-scale harbor environments. From the outset it was decided that multiple locations needed to be modeled, at varying levels of fidelity, to fully exercise the repeatability and scalability of this work. To accomplish this multiple models were created using tools that have greatly enhanced the ability to quickly generate 3D models.

Finally, after the components required to create a simulation were identified and four exemplar scenarios constructed, a detailed reporting capability was required to provide a means to formalize the results of the simulations in a fully integrated format.

D. DESIGN CONSIDERATIONS: INTENDED USER COMMUNITIES

The primary design consideration for this work was to ensure that the three targeted end users: Harbor Operations Support Staff, Harbor Security Planners, and shipboard force protection officers (FPOs) will hopefully be able to use this technology in a manner that serves their purposes. This required that the applications be powerful enough to handle complex simulations with the ability to perform a large number of replications of detailed experiments with design of harbor defenses and day-to-day assessment of operational risk. At the other end of the spectrum functionality had to be provided to allow war fighters, who have both less experience with simulation and less time to perform analysis, to ability to run simulations scaled to their limited area of focus for the purpose of planning and evaluating force protection plans. These design considerations ensured that users with varying levels of expertise and proficiency could benefit from the work in this project. Formally identifying the required skill sets for each end user is identified as future work.

E. EVALUATION OF RESEARCH APPROACH: NAVAL POSTGRADUATE SCHOOL M&S WORKSHOP

The Global War on Terrorism (GWOT) has brought defense against terrorist threats to the forefront of U.S. policy and strategy decisions. As a result, multiple research groups, companies, and organizations have worked on the problem of harbor defense. In May of 2006 the Naval Postgraduate School sponsored and hosted a workshop designed to create a forum for many of the practitioners in this field to come together and present their work to date, explore opportunities for collaborative work, and discuss the challenges and unsolved problems of harbor security.

This research approach of this thesis was comparable to the work currently being done in this area and unique in its use of the combination of DES, 3D graphics, and agent-based modeling. A total of 49 participants from 28 organizations discussed 17 presentations, providing an excellent overall survey of the state of the art in these critical areas. All presentations and a record of group dialog are included in the workshop report. (Brutzman, Blais & Norbraten 2006) This resource is available to U.S. government personnel and support contractors in CD or hardcopy form upon request.

The conclusions from this workshop identified the need for continued research and development of computer technology to aid decision makers, planners, and military personnel in the force protection community. (Brutzman, Blais & Norbraten 2006)

THIS PAGE INTENTIONALLY LEFT BLANK

IV. AGENT BEHAVIOR MODELING

A. INTRODUCTION

This chapter discusses the design and structure of agent-based behavior modeling for this thesis. The goal of creating larger harbor environments required that a multi-agent system (MAS) design be employed to allow for multiple behavior types. The goal of this agent design was to create a standardized approach that can be mapped to the event graph methodology of Simkit and Viskit. This chapter presents the theoretical framework for the agent-based approach that was taken. Implementation of this approach is discussed in detail in Chapter V.

B. EARLY APPROACHES TO BEHAVIOR MODELING IN THE ANTI-TERRORISM / FORCE PROTECTION DOMAIN

The predecessor to this thesis (Harney 2003) demonstrated an approach for developing agent-based design for the AT/FP problem domain. As part of this earlier work, Harney provides a discussion of the design considerations for the formal approach to MAS system design (e.g., MAS = {Environment, Objects, Agents, Relations, Operations, Laws}) introduced by Ferber (1999). Agent design for this thesis follows the same underlying fundamental principles.

The scope of Harney's work purposely limited the type of tactical behaviors to two subcategories: Defenders and Attackers. The environment was also limited to the confines of the harbor area. As a result, a stable framework was developed which created an end-to-end simulation of force protection scenarios.

As a logical successor to Harney's research, this project expands this agent behavior design and seeks to structure behavior definitions in a manner best suited for DES implementation.

C. SITUATED LOGIC PARADIGM FOR MULTI-AGENT SIMULATION SYSTEMS

To gain the most insight from a simulation agents should act autonomously and realistically based on defined rules. There should be no script that states how the simulation will progress or exactly what the agents will do. Instead, the simulation should evolve as agents interact with each other and the environment based on their defined behaviors.

Prior to implementing an agent behavior with event graph models it is desirable to sketch the desired rules that will drive agent behaviors with minimal concern for implementation. A common approach to defining these rules is Finite State Automata (FSA). FSA's, sometimes referred to as finite-state-machines have three major components: states, actions, and transitions. These components serve as a framework for creating a model of agent behavior and are widely used in the computer gaming, artificial intelligence (AI), and military simulation communities. FSAs are behavior based and decouple agent behaviors from transition logic. (Darken 2005) Figure 11 shows an example FSA diagram of a military patrol craft.

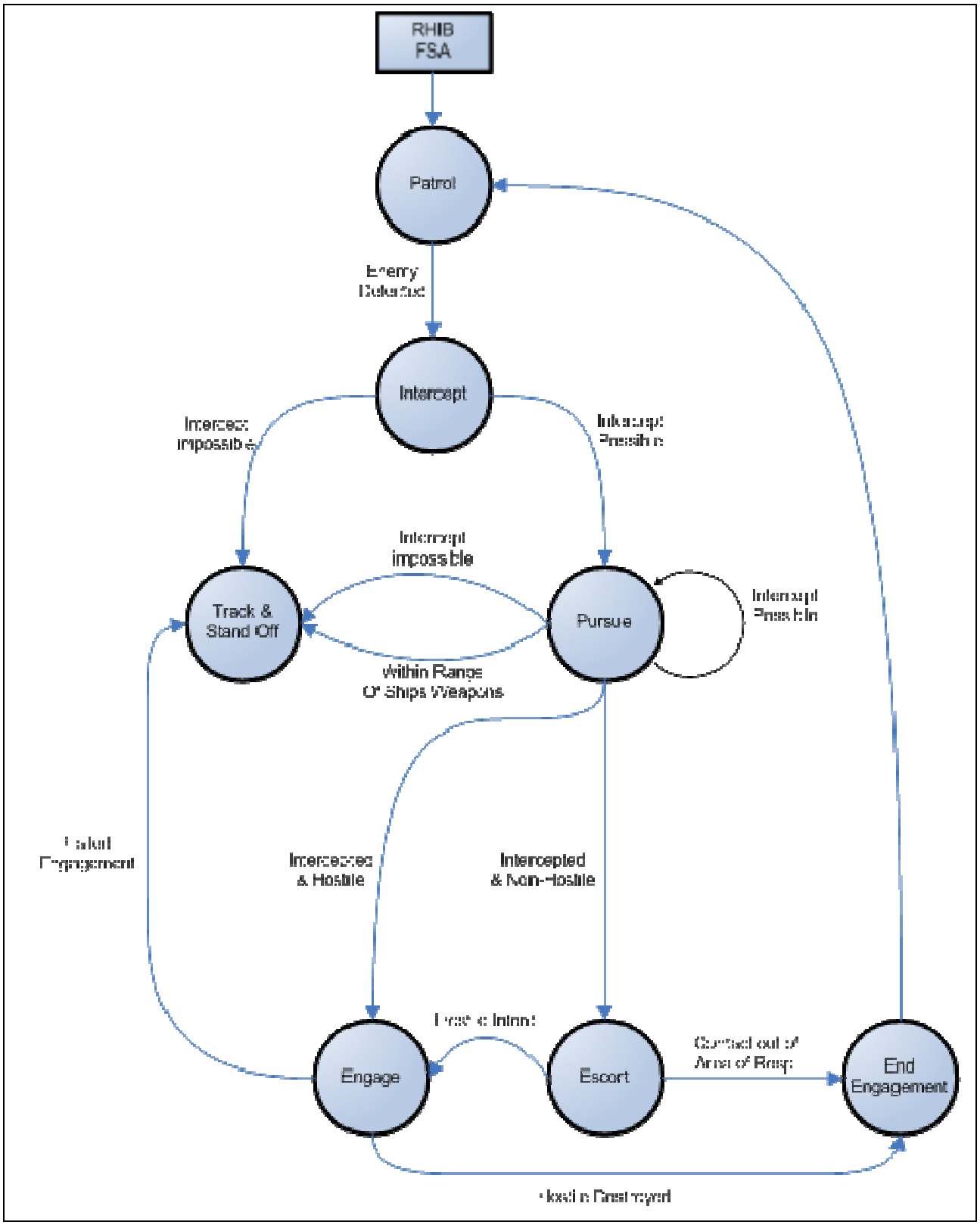


Figure 11. Finite State Automata diagram that outlines the desired behavior of a military patrol craft.

As seen in Figure 11, FSAs provide a high-level overview of an agent behavior. FSA graphical representation is not a complete representation and does not contain all of the information necessary to implement the model. This fact distinguishes FSA diagrams from DES event graph models, and it is important that the two are not confused. While behaviors or actions are captured in the states of an FSA diagram, they do not describe how that behavior should be implemented or what the behavior means. Surprisingly, it is this ambiguity that makes FSA diagrams a logical first step in behavior design. For tactical simulations, military personnel who are not involved in developing a scenario can still contribute to the development process by providing descriptions of tactical behaviors that can be mapped to an FSA diagram. The undefined actions and/or behaviors that each state represents can serve as a tool for conducting interviews with subject matter experts (SMEs) and help to ensure that no aspect of the behavior interactions are overlooked. All of these factors add to the utility of FSA diagrams as an early part of the development process.

D. AGENT MODELING FOR TACTICAL SCENARIOS

Before implementing the behaviors that are desired for an agent it is necessary to consider how generic agent behaviors (e.g., from Figure 11 'Patrol' or 'Intercept') are going to be implemented. This section discusses fundamental concepts that were used to create a behavior definition library that can also used to construct harbor defense scenarios.

1. Agent Relationships

Agents originally used for harbor defense in (Harney 2003) were limited in scope to defenders and attackers. By expanding the types of agents and objects used in the simulation it was necessary to create a formalized structure and organization for a behavior library. Figure 12 shows the basic organization of the agent behavior library. Agents and objects are sorted by type and grouped together under a common classification. For example a patrol craft is a friendly agent and a terrorist cell is a hostile agent. (Hiles 2002) notes the importance of establishing the relationship between agents as part of a MAS design. This structure gives agents affiliations to one another. This relationship can be leveraged to allow communication and shared information between

like entities. The library structure is for organizational purposes only. Agents themselves have no access to this information which might potentially affect a simulation (e.g., Friendly agents knowing who Hostile agents are). Agents are only exposed to information explicitly made available as part of the simulation, modeled after real-world communications so that realistic responses are possible.

Finally, while Figure 12 shows a number of entities with a real-world, intelligent counterpart there are also objects that would not be considered entities such as obstacles, and abstract objects such as nautical chart. Though these objects are not entities they are included in the library because they have event graph representations and are used by simulation agents in the environment.

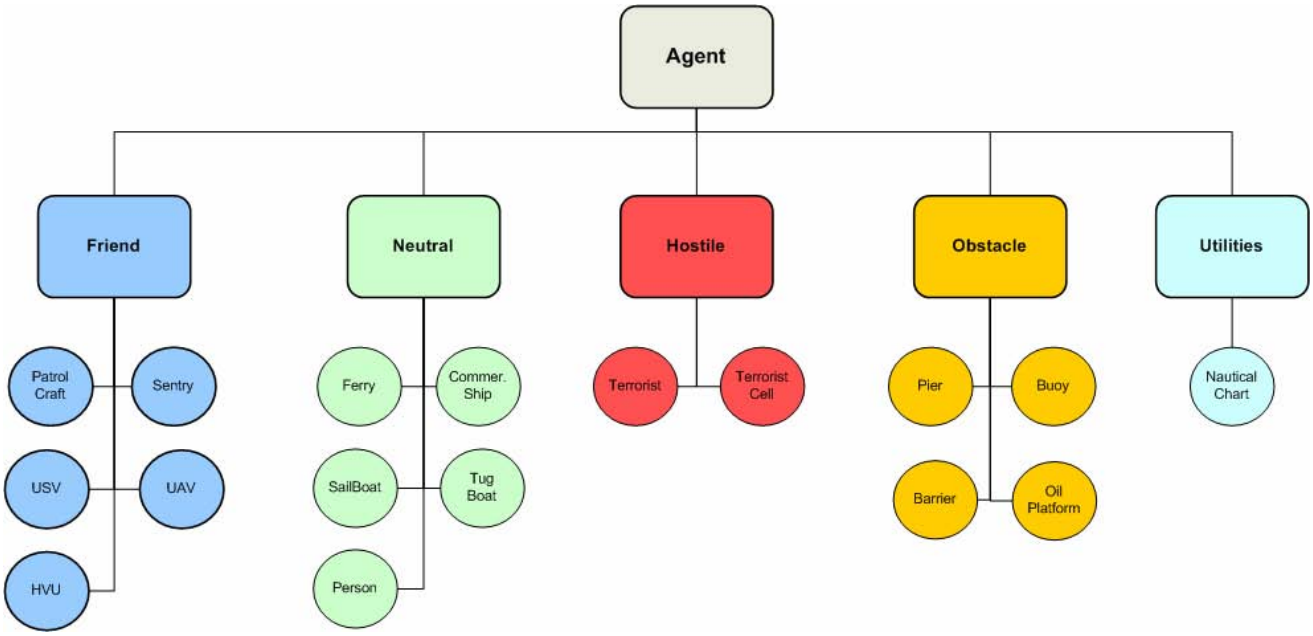


Figure 12. Diagram of the organizational structure of agent relationships and affiliations in the AT/FP project

2. Movement

Movement for this simulation follows the same patterns originally developed in the Simkit API. (Buss & Szechtman 2006) provide an excellent discussion of basic movement principles for DES complete with examples and source code. For the purposes of this discussion the basic principle is sufficient. An entity starts at a given position and proceeds to a destination at a specified velocity. At the end of each move an

'End Move' event occurs and schedules a 'Start Move' event without delay. This pattern repeats as long as the entity has another desired destination at the end of a move. Figure 13 shows a simple representation of this pattern.

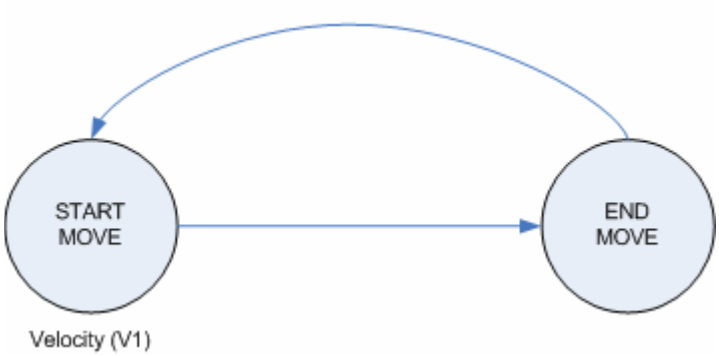


Figure 13. A diagram of the basic movement process for a single entity using Simkit.

Managing the movement for an entity is handled by a Mover Manager. Originally developed in the mover libraries of Simkit, mover managers define the starting and ending positions for a given entity. Diskit has expanded upon the Simkit mover manager library to allow movement in three dimensions, however the same mathematical principles apply. Table 1 contains some of the mover managers from the Diskit API.

Path Mover Manager	Moves an entity through waypoints that form a path.
Intercept Mover Manager	Moves an entity from its starting position to an intercept position. After intercept is complete the original mover manager is restored.
Avoidance Mover Manager	Moves an entity from its starting position to a point which avoids another entity or object based on a defined avoidance range. After avoidance is complete the original mover manager is restored.
Zone Mover Manager	Moves an entity utilizing either probability zones or A-Star Search Zones (discussed below)

Table 1. Table of Mover Managers used to move agents through the environment in the AT/FP project.

A Path Mover manager uses a series of waypoints to move an agent along a specified path. Figure 14 shows an example path that represents a series of waypoints for an entity to follow.

When an entity reaches a destination waypoint the Path Mover Manager sets the entity's current location as the new starting position and provides the next waypoint in the path as the updated destination. Figure 15 shows a model of a Rigid Hull Inflatable Boat (RHIB) using a Path Mover Manager to follow the path depicted in Figure 14.

Figure 14. Depicts an example path (in red) that is the desired route of an entity in the Bremerton, Washington scenario

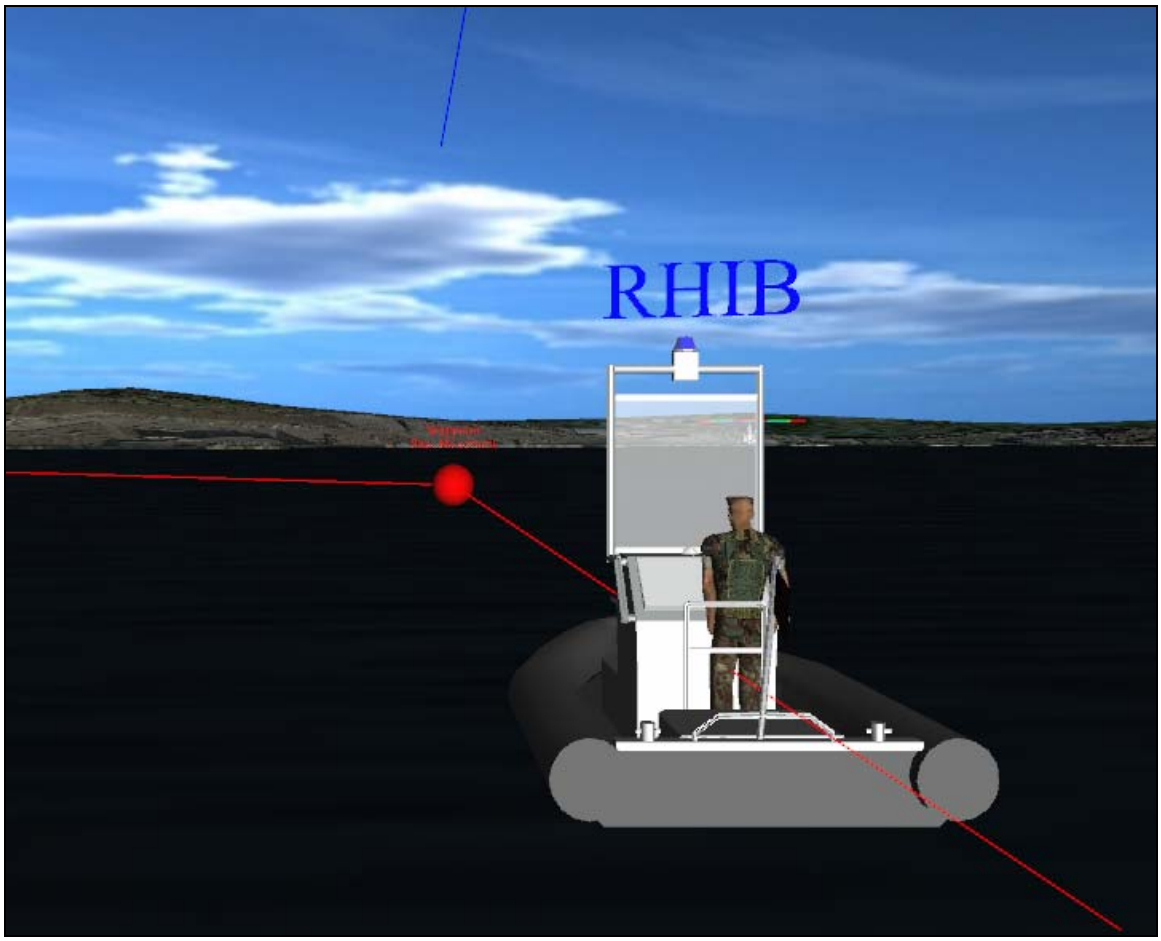


Figure 15. Depicts an entity using a Path Mover Manager to navigate waypoints along a path in the Bremerton, Washington, scenario.

Path Mover Managers are useful for a limited number of entities. It is not desirable to create a set of fixed waypoints for an entity to follow in most situations. If a Path Mover Manager is utilized, an entity follows the same series of waypoints for every simulation run. In this project, the only application of a Path Mover Manager is for harbor ferry traffic which follows a set, pre-defined route.

The Zone Mover Manager (ZMM) is the primary mover manager utilized in this project. This mover manager was originally created to address the problem of simulating patrolling behavior. In the case of a patrol craft, real-world personnel conducting patrols are not given a series of waypoints and instructed to patrol according to a predefined path. Rather, they are assigned areas of responsibility and are expected to cover those areas during their patrol.

The ZMM uses a waypoint creator to assist in the waypoint-generation process. A waypoint creator is used when the method of generating waypoints is more complex than utilizing a predefined set of points as is the case for the Path Mover Manager.

Figure 16 shows a bird's eye view of a harbor with three zones defined. These zones represent areas where the agent is responsible for patrolling. As the ZMM moves its agent, it selects waypoints from one of the zones that are considered part of the agent's responsibilities. The frequency with which a waypoint is generated from each zone is also controllable. This control is achieved by expanding upon random plot generation theory as outlined in (Buss 2006). Each zone has an associated probability which defines the frequency with which a waypoint is generated in a zone.

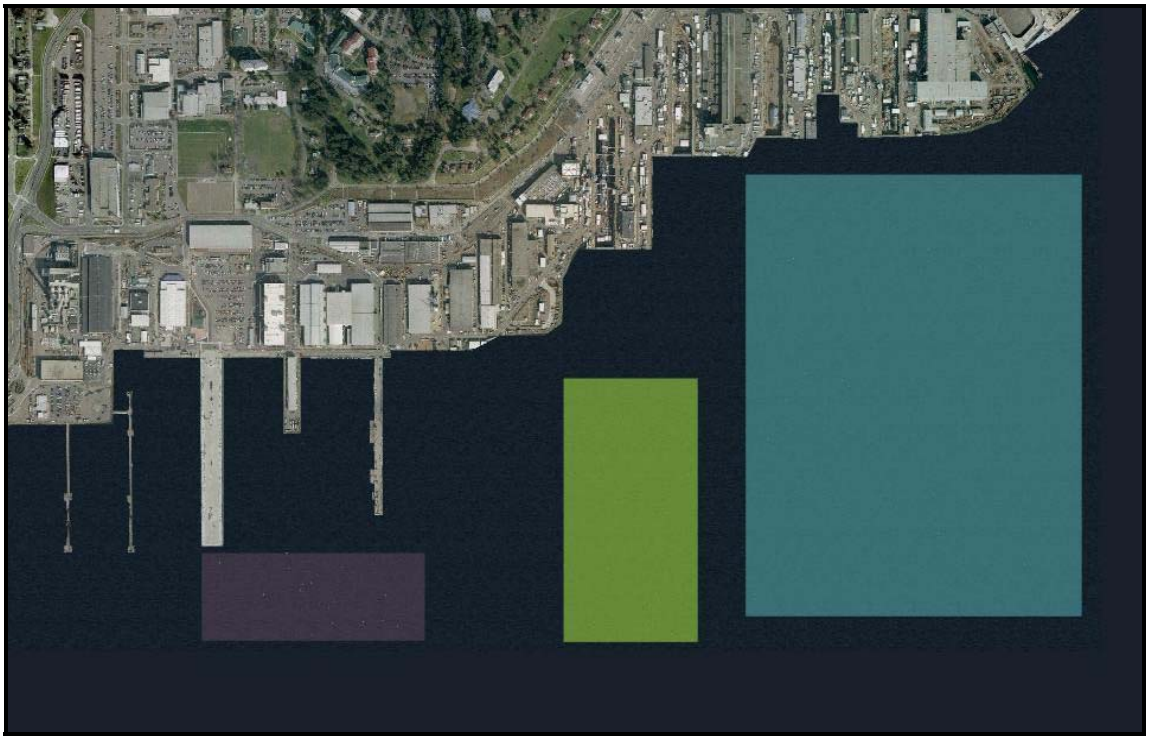


Figure 16. Visualization of three patrol zones in the Bremerton, Washington, scenario.

The ZMM uses this probability distribution function to select the next waypoint. Random plot generation creates a waypoint anywhere within the bounds of a zone.

Figure 17 shows random waypoints generated in two zones using this concept. One thousand waypoints were generated and plotted in this example. The larger zone

had an associated probability of eighty percent and the smaller zone twenty percent. The density of the distributions corresponds to those probabilities.

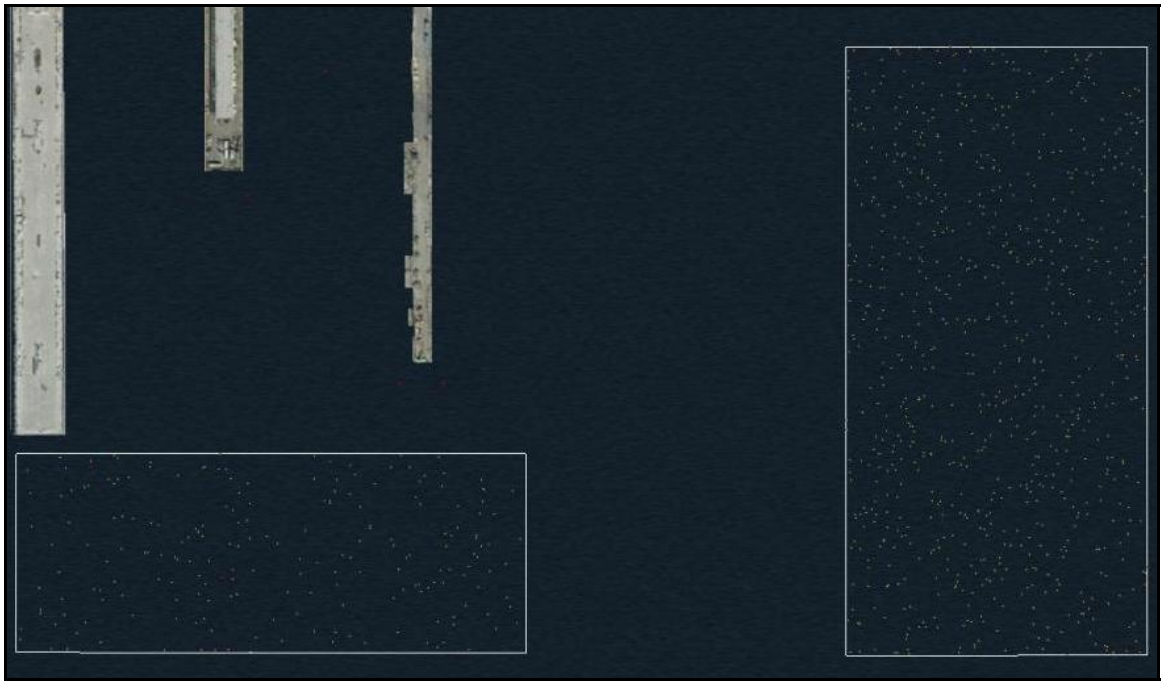


Figure 17. Depicts a waypoint distribution plot using random plot generation theory

It is important to note that this probability is not directly proportional to the amount of time an agent will spend in each area. For example, if an agent was patrolling two zones and the zones were located twenty miles apart the majority of the agent's time would be spent transiting between the zones. Adjustments to a zone's size, probability, and proximity to other zones can achieve desired durations spent in an area.

The movement pattern described above adds to the utility of the simulation as a whole. For any given simulation run, the analyst does not know exactly where the agent is going to start or travel to next. While the analyst can specify the probability with which the agent might reach a specific destination within a zone, they are unable to specify the exact waypoint within the zone that the mover manager might select. Such ambiguity is representative of real-world patrol assignments since it is impossible to predict with certainty the exact position of defensive forces at the time of an attack.

Zone Mover Managers can also be used to implement A-star (A*) search algorithms. A-star search is a path-optimization algorithm that uses a heuristic path-cost metric to find the shortest distance between two points. Outlined in (Darken 2005) and (Russell & Norvig 2002), A* can be used by agents that need to determine the shortest logical path to take between a series of points. Figure 18 shows a simple diagram to illustrate the basic principles of A*.

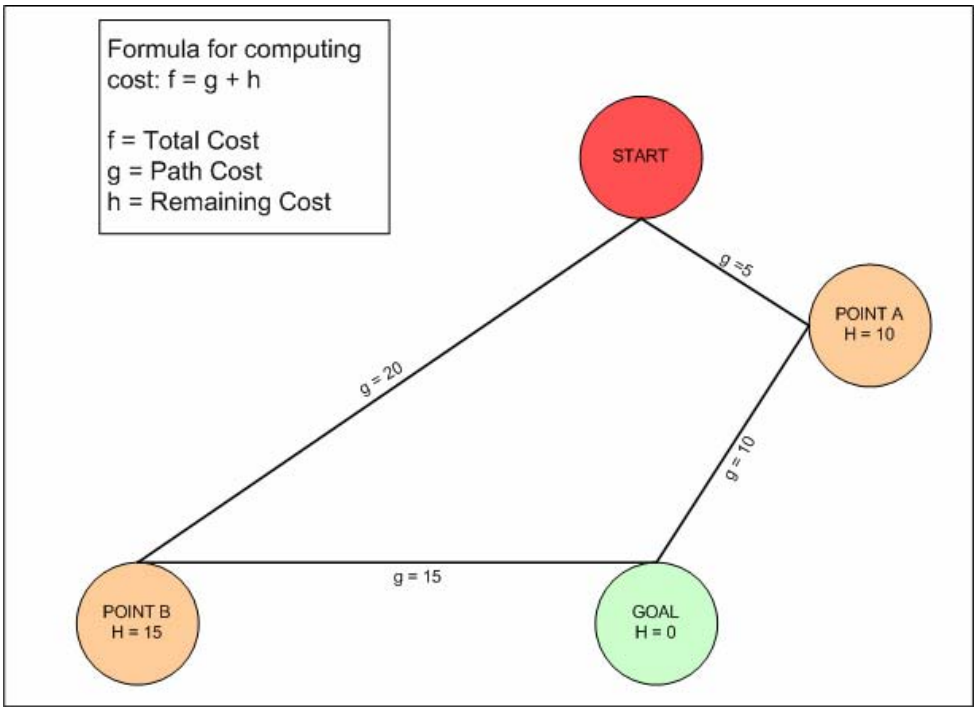


Figure 18. A simple path example illustrating the A* search framework

In Figure 18 the red node labeled Start is the starting position, and the desired destination is labeled Goal. In this example there is no direct path between the start and goal nodes. Therefore an agent must be able to decide which intermediate node, either Point A or Point B to go to on its way to its final goal. A* seeks to optimize the path selected which for our purposes means the shortest distance between two points. This is accomplished by utilizing the formula f = g+h where f is the total cost for a move from a starting node to the next node, g is the cost of the path between two nodes, and h is the remaining path cost to the goal should the agent decide to move to the node in question. In our example both intermediate points provide a path to the goal node but taking a path via Point A is considerably shorter. Applying the formula an agent at the starting node

would compute that the total cost f of moving to Point A would be fifteen. Point B has a total cost of thirty-five. Using the algorithm the agent would choose to transit to its destination by moving to Point A first which is the desired result.

The heuristic value (h) can be modified for various applications. If for example an agent was trying to avoid being detected then the remaining cost (h) could be a function of the remaining path cost from a node plus an additional cost i.e., probability of detection, for that node. In our example if the probability of being detected by transiting through Point A was 45% then the total cost for transiting through Point A could be fifty-five, the original cost plus an added cost for being detected. If Point B had a 0% probability of detection then the cost of that path would be less and the agent would move in a manner that optimized its path and minimized its probability of detection. For this project A* was implemented only to optimize path planning. Expanding the heuristic function for various behaviors is discussed further in Chapter IX.

An A* implementation was added to the Diskit API to facilitate this path finding approach. Rather than having simple nodes representing path choices as in Figure 18 rectangular zone geometry objects were used. Creating the cost values of the A* structure was accomplished by using the three dimensional coordinates of zone geometry objects. Figure 19 shows a path structure similar to that of Figure 18.

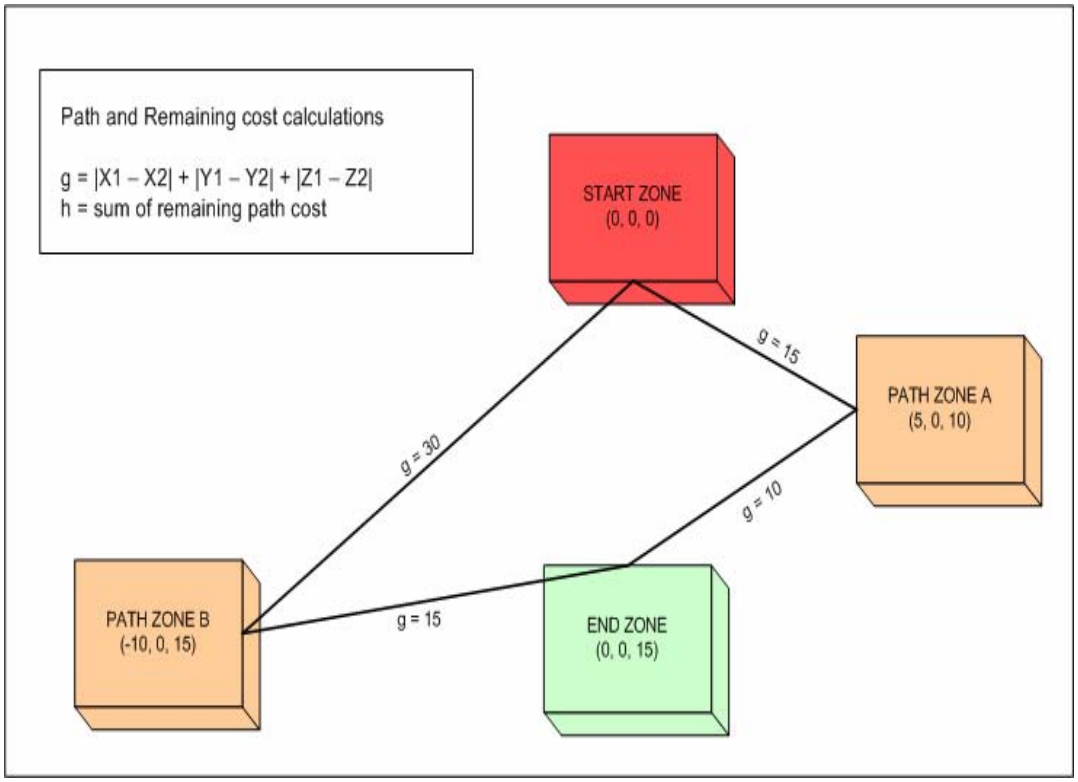


Figure 19. Depicts the A* search structure used in this thesis with three dimensional zone geometry objects

As shown in this example path cost between nodes (g) was calculated by summing the absolute value of the differences between the X, Y, and Z center point coordinates of two zones. This implementation automatically calculates the cost values for a complete A* map based on the defined zones of the map, the relative distance between each zone, and their proximity to the goal zone. An example overlay of a complete map structure in a harbor environment is provided in Figure 20.

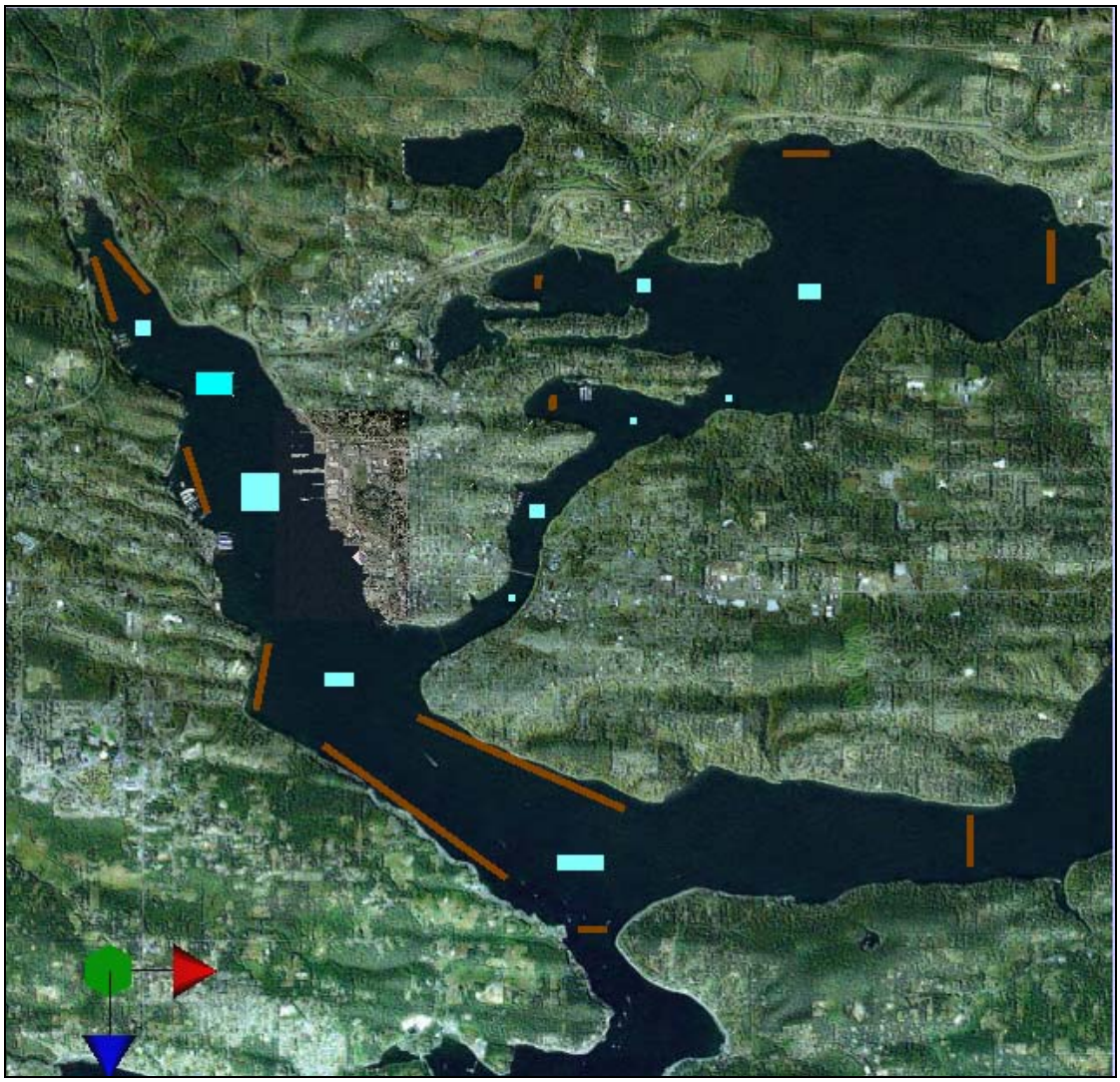


Figure 20. Depicts the A* zone implementation overlay of the Bremerton, Washington, harbor

The light blue and brown rectangular shapes shown in Figure 20 represent an example A* map that can be used by agents to optimize path planning in a large environment.

Zone Mover Manager objects that utilize the A* search algorithms in Diskit provide their movers with a series of waypoints that are generated based on the most efficient path between two zones. By using zone geometry objects the general path can be identified by a series of zones but the exact waypoint selected inside of each zone can be generated using random geometric plot theory discussed earlier. The resultant

behavior is that agents move in an efficient manner but the exact path is not likely to be the same between simulation runs.

Mover managers can be changed dynamically while the simulation is running to allow for a variety of movement types. While the primary mover manager for agents is the ZMM, if an agent needs to avoid an obstacle or another contact it will use the Avoidance Mover Manager (AMM). Likewise the Intercept Mover Manager (IMM) is used when intercepting a contact. The velocity calculations required to perform these movements, which are modeled after those found in the Simkit, are handled by the Movement Calculator utility class of the Diskit API. The AMM and IMM use the Movement Calculator to generate waypoints instead of a Waypoint Creator. The combination of mover managers discussed in this section enables agents in this project to perform the movements required in the simulation.

3. Sensors

In addition to an agent's ability to move, an agent must also have a means to sense its environment. The Diskit API includes sensor objects for this purpose. The sensors used for this application are simple, spherical sensors. These sensors are a 3Dl implementation of Simkit's Cookie Cutter Sensor. A Cookie Cutter Sensor is defined by its radius and has the ability to detect any object that moves within its range. Figure 21 illustrates a simple Simkit Cookie Cutter Sensor detection scenario.

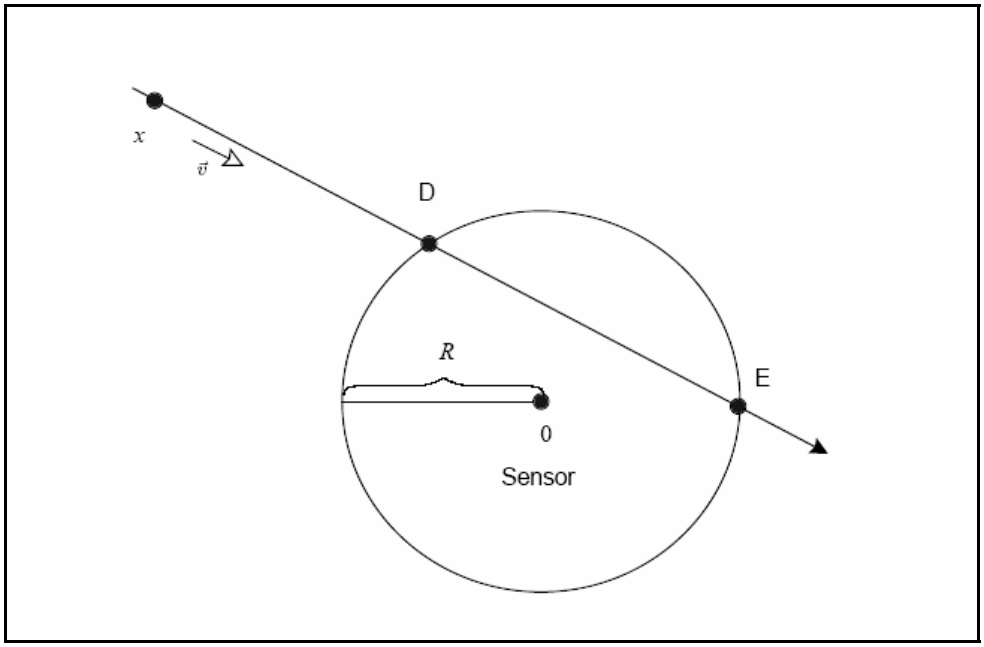


Figure 21. Depicts an illustration of the basic Simkit cookie cutter sensor scenario. (from Buss & Szechtman 2006)

This example shows a sensor of radius ‘R’ depicted by a circle and a mover ‘X’. The mover is traveling along a straight line at velocity ‘V’. Simkit uses this information to calculate the times at which the mover will enter and then subsequently exit the range of the sensor. This model is followed for Sphere Cutter Sensors in Diskit with the calculations performed in three dimensions. In Figure 21, points ‘D’ and ‘E’ show when the mover will enter and exit the sensor range. For DES these two entry and exit intersection points are captured as events. A Detection event is scheduled for the moment when the mover is predicted to enter the sensor range. Likewise, a Un-Detection event is scheduled for the moment when the mover is predicted to exit the sensor range. These events are added to the event list and scheduled to occur at times which correspond to the calculated enter/exit times.

Such a process is repeated in a pair wise fashion for each sensor and mover combination. Thus the entire set of predicted interactions can be computed and predicted for a collection of entities and sensors, as long as each maintains constant course and speed. Should any entity change their velocity vector, recalculating all sensor enter/exit points restores the schedule of interactions.

A similar approach can identically be used for scheduling potential collisions between entities. Thus a powerful modeling and simulation paradigm is achieved: pair wise entity sensor detections and object collisions can be implemented via a DES event queue, rather than requiring frequent time-step computations for each possibility. This is a major accomplishment which greatly speeds up the computational overhead required for an adequate-fidelity kinematics-based physical simulation. A complete discussion of the mathematical concepts and a number of simple examples can be found at (Buss & Szechtman 2006).

The agents in this project can have multiple sensors. To illustrate this concept we will examine the sensors of a Rigid Hull Inflatable Boat (RHIB), which is an instance of a Military Patrol Craft behavior. In the case of the RHIB, the agent has a visual sensor and a collision avoidance sensor. Figures 22 and 23 provide a visualization of these two sensors. Each entity establishes a listener connection to its sensors which allows the agent to be notified by its sensor when detection and un-detection events occur. This notification is achieved by the scheduling of Detection and Un-Detection events. As the names of the sensors imply the visual sensor is used to represent the visual range of a human on the RHIB. The collision sensor is used to alert the agent if another agent or object is too close and requires attention to avoid collision. How the agent reacts to sensor detection events is defined in the event graph model. Additional information on the implementation of sensors can be found in the DES with Simkit documentation (Buss 2001).

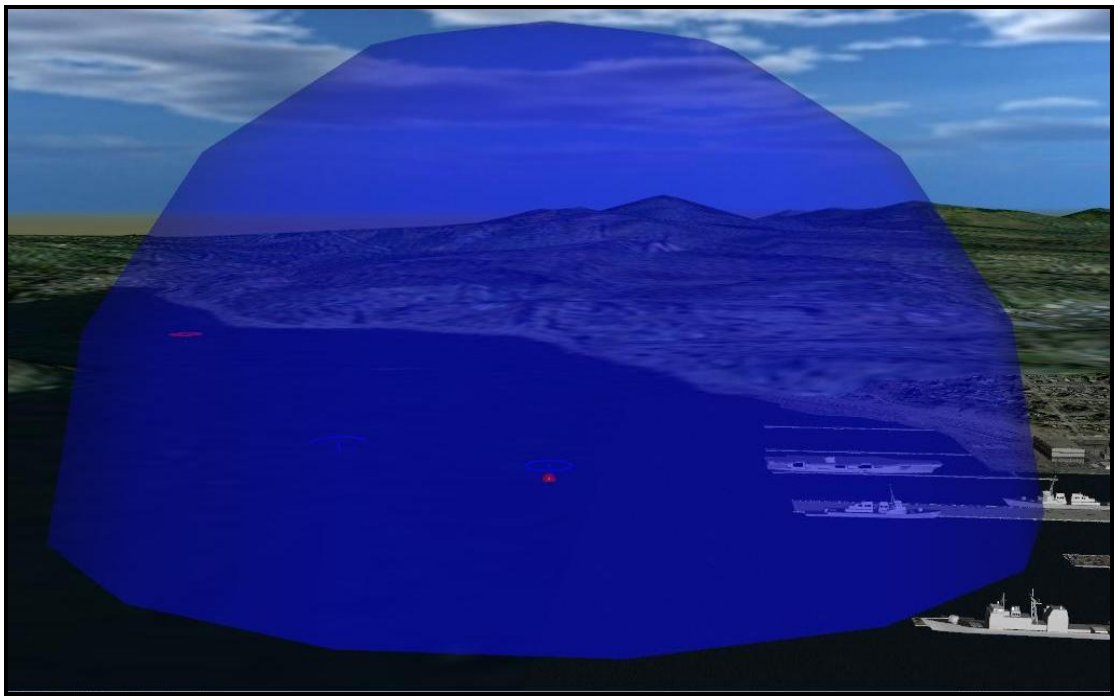


Figure 22. A visualization of a visual sensor for a human on a military patrol craft.

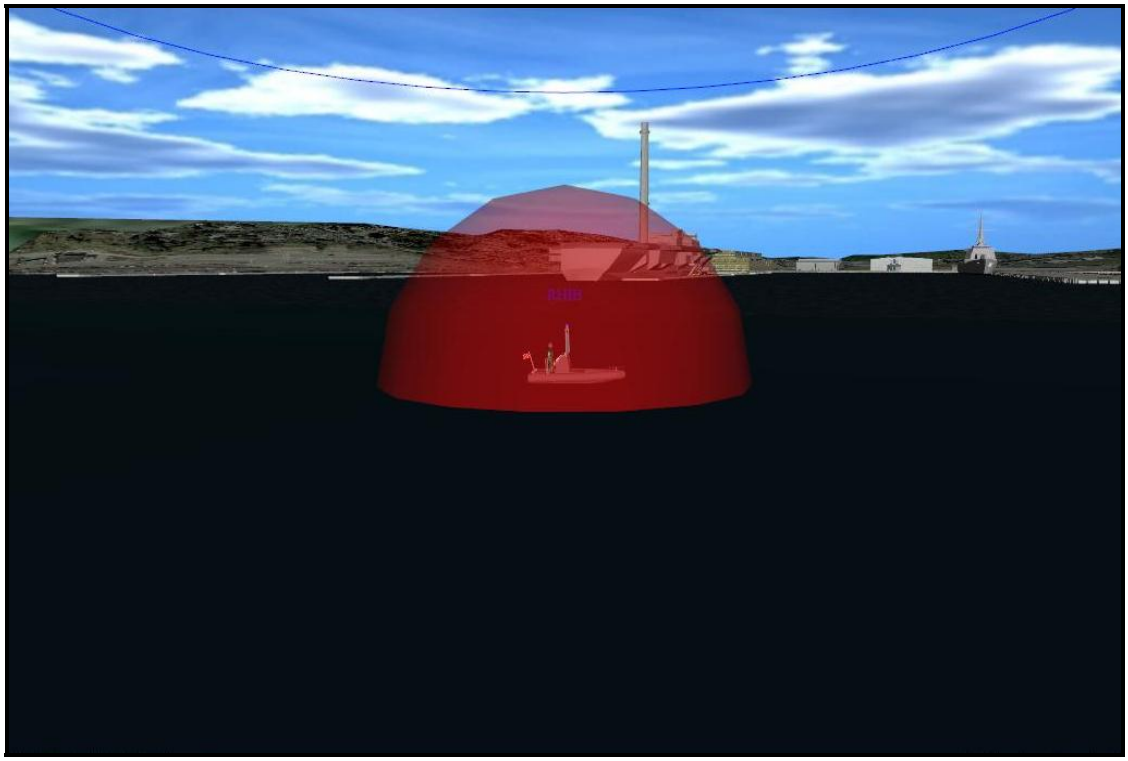


Figure 23. A visualization of a collision sensor for a human on a military patrol craft.

The Sphere Cutter Sensors used in this project are not exact representations of the nonlinear real-world sensors they are designed to simulate. Nevertheless reasonable

linear approximations for sensor response are available and commonly used for tactical solutions. Thus this approach has merit. Work on higher-fidelity sensors is being conducted and is also identified as recommended future work.

4. Communications

The majority of tactical situations are greatly dependent on the ability of individuals to communicate. Harbor defense is no exception. In a typical harbor environment there are a number of independent participants that must be able to communicate information effectively in order to execute any AT/FP plan. Failure of (or an interruption to) the normal lines of communications creates a potentially vulnerable tactical situation for the entire harbor.

To simulate this real-world dependency on communication between participants, radio-communication objects were implemented in Diskit and used extensively in the event graph models. To formalize the implementation of radio communications, a template was created which outlines the information required to send and receive radio messages. This framework is outlined in Table 2, which lists the required components for radio communication objects.

Component	Description
Channel	An integer value that represents the communication channel number.
Sender	The entity that is sending the message.
Recipient/Recipients	The intended recipient or recipients of the message. Only intended recipients receive the message.
Context	The agent that the message is about, if one exists.
Message	The text of the message that conveys the purpose and meaning of the message. This text is used for display purposes only. The processing of radio communications is handled by message type.
Message Type	A standard String representation of what the type of message being send. Some examples include: Warning Report, Query, and Query Response.

Table 2. Table of the required components for a radio-communication object.

Standardizing the message format for communications was required so that behavior event graph models could handle a variety of message types in a consistent and repeatable manner.

By implementing radio communications, analysts have the opportunity to evaluate the impacts of failed or unreliable communications. The consequences of compromised communication networks can also be examined. More importantly, the realism of models reflecting real-world interactions can directly mirror the progress of teamed-entity interactions in the real-world.

Messages are sent and received through the passing of Radio Communication objects from one entity to another. The specific event graph model used to implement this transmission is discussed in Chapter V.

Figure 24 provides a display of communication circuits as they are being used during a simulation run. The figure below was created by simply outputting the message text of radio traffic as it was transmitted. The ability to track and monitor radio communications in this fashion enhances the model-development process and allows a user to verify that the desired flow of communications is occurring as expected.

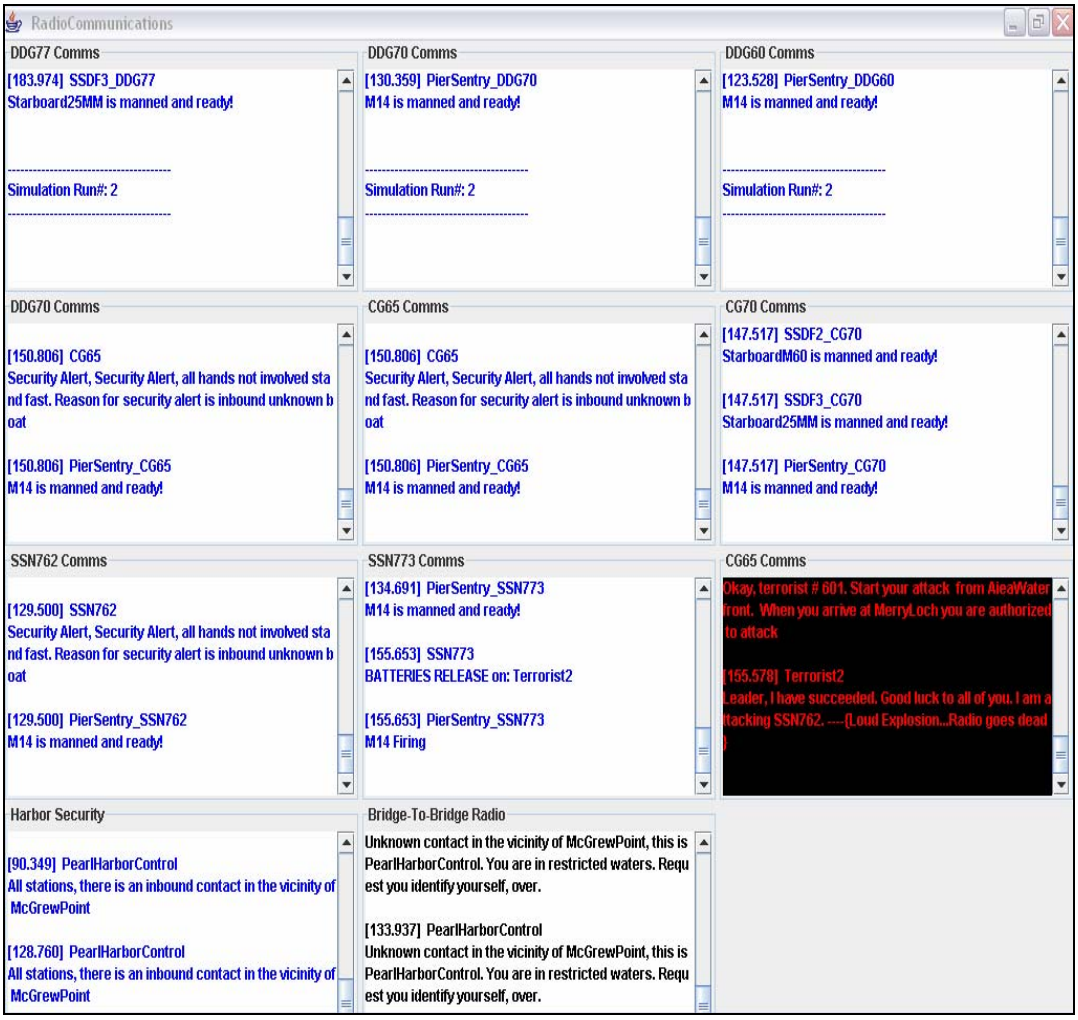


Figure 24. A multiple communication display panel that shows communication between agents on eleven different radio circuits.

5. Harbor Environment

(Hiles 2006) and (Harney 2003) both discuss the importance of identifying and defining the environment for a MAS. Expanding the original AT/FP environment to include areas outside of the harbor and above the water surface required the creation of a repeatable methodology for representing that environment. Utilizing the concepts discussed in this chapter, two important components were implemented: a nautical chart and generic obstacles.

Figure 25. A visual representation of the Nautical Chart used by agents in the Pearl Harbor., Hawaii, scenario.

The nautical chart object is an abstract representation of the harbor water ways and is formatted as an A* search map. Figure 25 illustrates another A* search zone map implementation.

The nautical chart is comprised of two sections: Waterways that can be navigated (shown in light blue), and Perimeter zones which can be used as starting areas for attackers (shown in purple). The A* implementation discussed earlier is applied only to the inner water zones that can be navigated. The outer zones are used by hostile agents to select a starting position. By providing a nautical chart representation all waterborne entities have the opportunity to navigate throughout the harbor using this object.

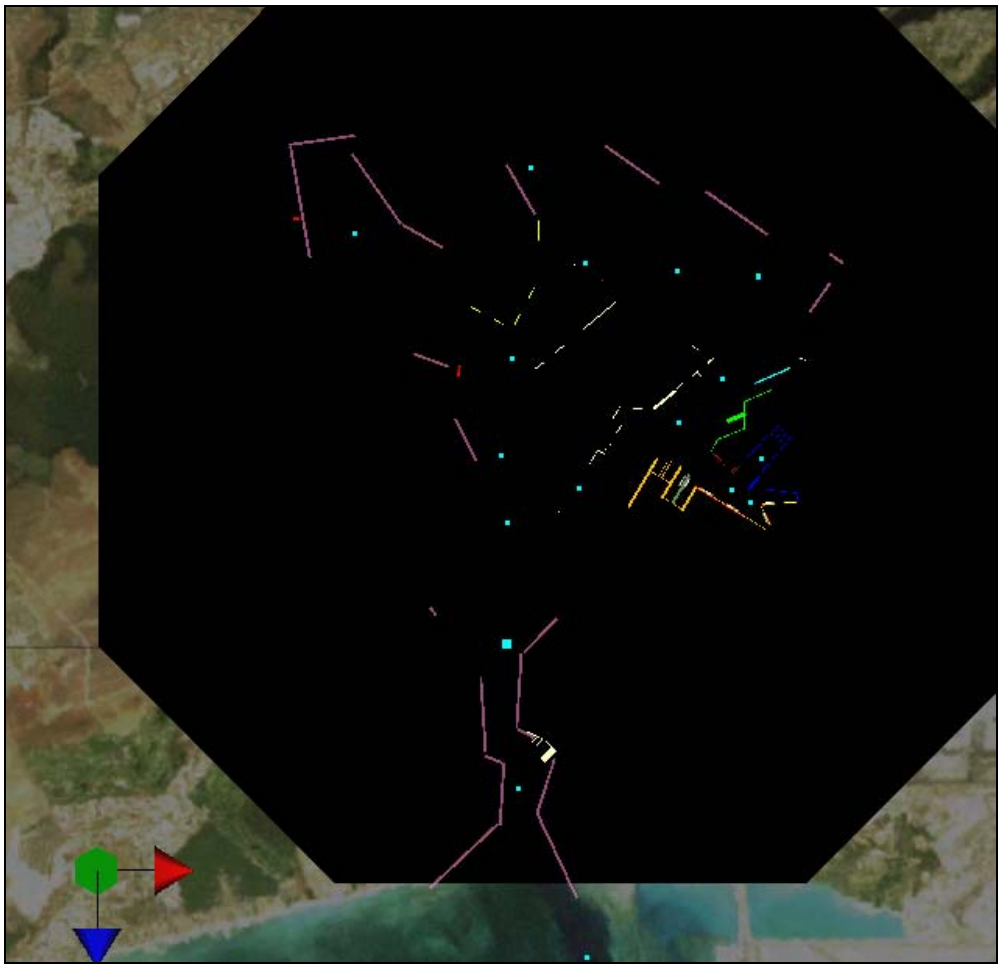


Figure 26. Abstract objects of the Pearl Harbor scenario with the terrain imagery removed.

In addition to the water environment there are a number of physical objects in a harbor that must have a representation in order for agents to interact with their environment. In this context those objects are buoys and piers. Figure 26 shows the same environment as Figure 25 but without the terrain imagery.

With the terrain removed it is easy to see the physical objects that are being represented. Creating these objects was greatly simplified by using the Vizx3D authoring tool. The real time display of 3D graphical content was invaluable to simulation design and served as a guideline for scenario development. A tool used for this purpose should have the ability to fully navigate and manipulate the environment. Once the objects were identified they were implemented in a manner that would allow simulation agents to interact with them. A full discussion on environment implementation is provided in the following chapter.

E. SUMMARY

This chapter discussed the underlying agent based modeling principles that were used to create AT/FP scenarios. A variety of agent-based modeling principles were used to provide agents with the ability to move in an environment, react to their surroundings and other agents, and to communicate in a manner representative of their real-world counterpart. The agent-based modeling principles used for this project are widely used and well documented and have been applied to many domains. The following chapter will discuss in detail how these high level concepts were mapped and implemented in event graph models.

V. DISCRETE EVENT SIMULATION AUTHORING WITH VISKIT

A. INTRODUCTION

This chapter will describe how the agent design pattern and simulation environment outlined in Chapter IV was implemented using Viskit. Specific attention is given to describing the specific approach undertaken to merge DES methodology with the desired agent-based simulation structure. Using Viskit, the resultant agent-based simulation combined the movement, detection, and communication elements of Chapter IV to create a DES for AT/FP problems. Through this discussion the many issues resident in modeling agent behaviors with DES will be examined with a discussion of the approach taken using the Viskit interface.

B. VISKIT/DISKIT AGENT INHERITANCE STRUCTURE

The Viskit GUI allows for the creation of event graph models that have the ability to autogenerate executable source code. As a result, users can quickly make complex event graph models without worrying about the line-by-line coding issues present in hand coded simulations. A potential difficulty with this robust authoring capability is creating event graphs that are so complex that they are hard to understand and follow.

This problem was quickly recognized in early thesis efforts in agent-based modeling. Figure 27 shows an early attempt at creating an agent behavior model for a waterborne patrol craft. In trying to capture all of the events required for a complete behavior definition, the event graph became large and difficult to understand.

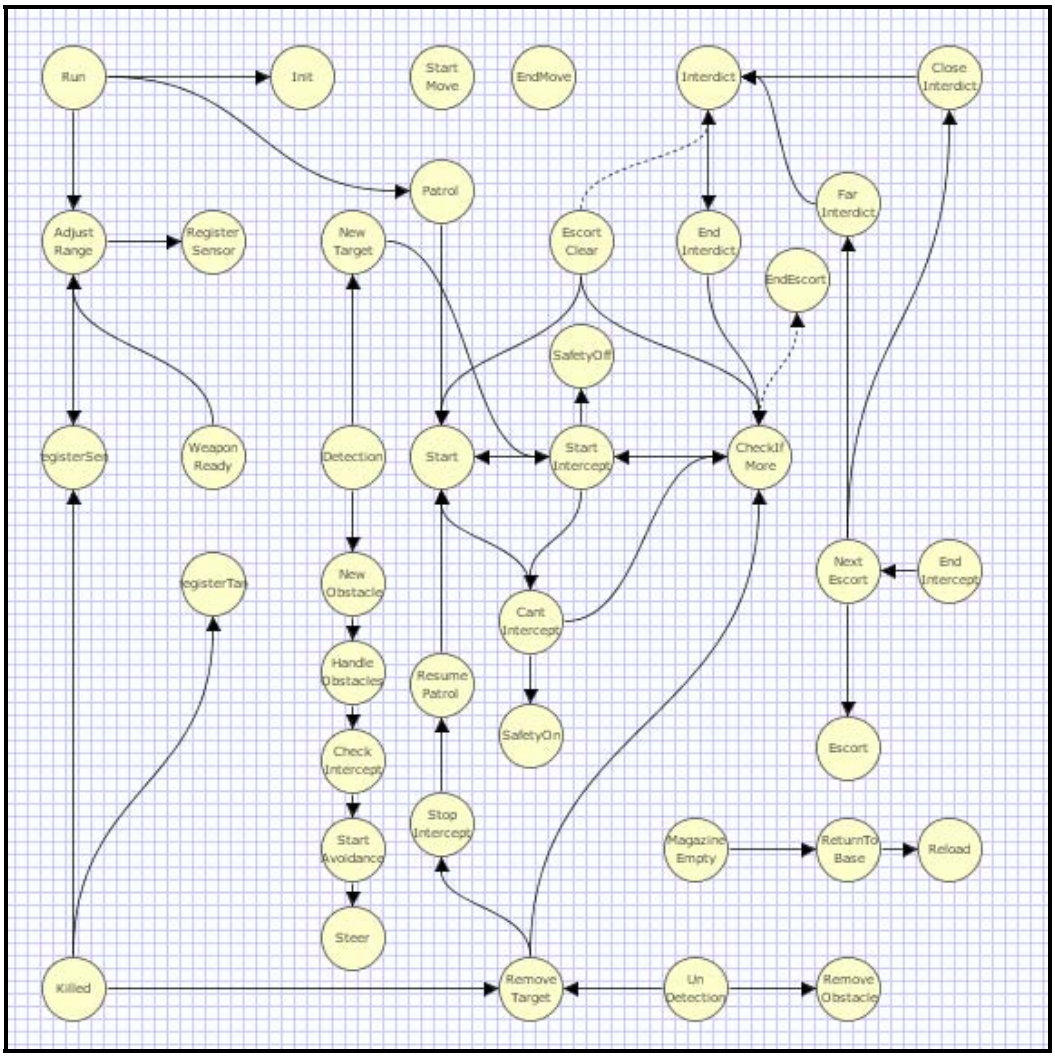


Figure 27. Defender event graph that demonstrates the complexity of trying to include all agent information in a single event graph.

Although somewhat unwieldy, the event graph model in this example is complete and there is nothing wrong with the implementation. Through the process of development, a number of event-graph design patterns were discovered for organizing relationships for interactions among a large number of entities.

The Java programming language allows inheritance which can be used to implement these commonalities in a logical manner. Inheritance as described in (Wu 2004) provides the ability to design multiple entities that are different but also share many features. By implementing inheritance, a parent or super class is created that combines all of the common features between entities. Other classes can then be created as extensions of that class while maintaining access to the common features of the super

class. This concept is used throughout the Simkit/Diskit APIs. In the context of agent behavior modeling the justification for this approach is apparent. All moving entities, regardless of type, have common requirements for participating in the simulation. The most obvious of those is the required ability to move. In Figure 27 a number of events are included just to facilitate movement. By creating event graph models that handle subsets of agent behaviors the actual behavior event graph becomes less cluttered, more focused on entity specific behaviors, and easier to debug. The remainder of this section explores the inheritance framework for this project.

1. SimEntityBase

SimEntityBase is a fundamental component of Simkit based simulation entities. For the purposes of this discussion it is sufficient to note that all entities, objects, and classes must extend SimEntityBase at some level in order to be used by the simulation. Further discussion on SimEntityBase is provided in (Buss 2004). In general SimEntityBase provides the simplest required structure for a simulation entity

2. DISMover3D

DISMover3D is a parent class in Diskit that, as the name implies, provides the minimum required components for a mover for a 3D simulation that is exposed to the DIS protocol. Figure 28 shows the event graph representation for this object. This class only provides an entity with the ability to move in the simulation. In this event graph a number of events, originally resident in the defender example, can be moved to the parent class. It is noteworthy to mention that although an event graph representation is provided all of the parent classes discussed in this section are written in Java source code. Currently, event graph models created in Viskit do not have the ability to inherit from each other. This feature is identified as future work.

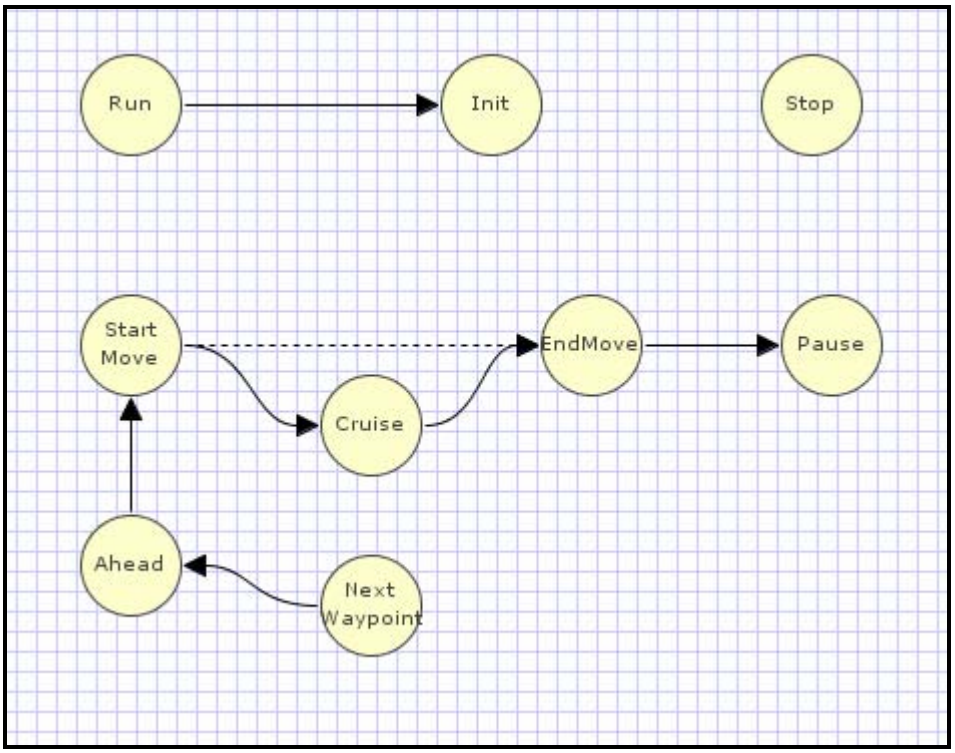


Figure 28. The DISMover3D event graph. The simplest form of DIS enabled moving entity

3. SMALMover3D

SMALMover3D is similar to DISMover3D in that it provides an entity with the basic requirements to participate in a simulation and move. This mover however exposes an event graph model to the Entity Definition portion of the SMAL schema. This additive feature allows a structured formalized parameter set that can be used in an event graph. The Entity Definition outlined in (Rauch 2006) not only specifies parameters but also declares default values for these parameters. SMAL was implemented in the Diskit API as a series of interfaces that match the structure of the Entity Definition schema. These interfaces identify all of the required parameters for a SMAL based entity. Figure 29 provides a diagram outlining the SMAL implementation. Unlike DISMover3D, SMALMover3D does not directly extend from SimEntityBase. Rather, there is an intermediate class called SMALEntity.

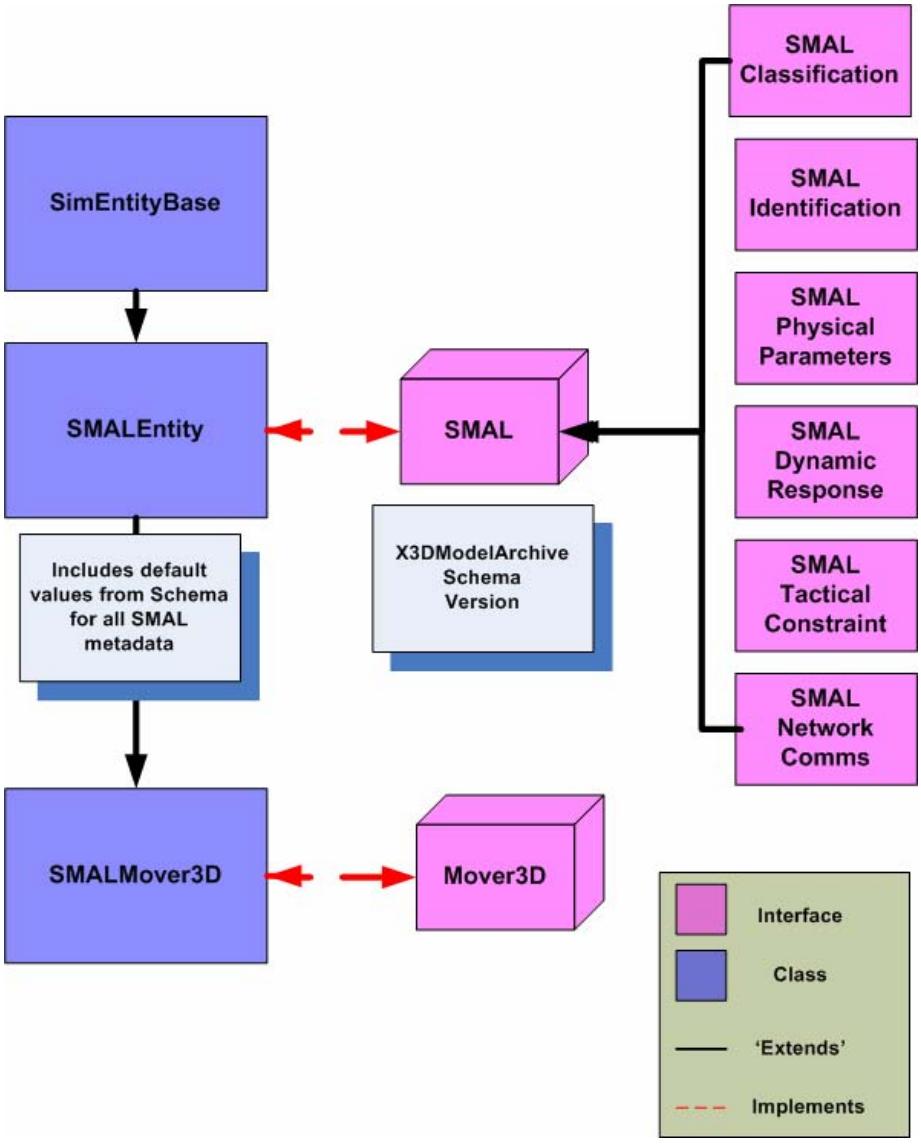


Figure 29. Diagram of the SMAL implementation in the Diskit API.

The purpose of the SMALEntity class is to ensure that all default values are initialized for every SMALMover3D object. For example the SMAL Entity Definition identifies maximum speed as a required parameter. The SMALEntity class initializes this parameter and sets the value to the corresponding XML schema default of zero. Agents that extend SMALMover3D can override these values with a user specified value. This design was chosen to allow users to fully utilize the SMAL Entity Definition schema without requiring that every value be implemented by the user.

A structured, detailed parameter set is not the only advantage of using SMAL. The primary motivation for exposing SMAL to event graph models is to allow for the auto-generation of Viskit simulations by tools such as Savage Studio. Further discussion of this concept is provided in Chapter VIII.

4. Mover3D

Perhaps the most important component of the entity structure in this project is the Mover3D interface. All objects that agents will interact with must implement the Mover3D interface. The primary reason has to do with object detection. Sensors in this project are designed to schedule Detection and Un-Detection events of Mover3D implementing classes only. The naming convention of both DISMover3D and SMALMover3D explicitly declares that these classes implement the Mover3D interface.

As with the SMAL interface, the Mover3D interface identifies the minimum requirements for a 3D moving object. Since all agents and objects in the simulation implement this interface all events that deal with object interaction treat objects as generic instances of a Mover3D. This design allows DISMover3D and SMALMover3D objects and agents to interact in the same simulation.

5. Force Types

Chapter IV presented the structure of the agent-behavior library. Figure 30 shows how inheritance is used within that structure. Again the goal remains to combine as many common features between like entities as possible, in order to allow event graph models to contain only the details of a unique behavior. Such inheritance is accomplished using intermediate classes called force types.

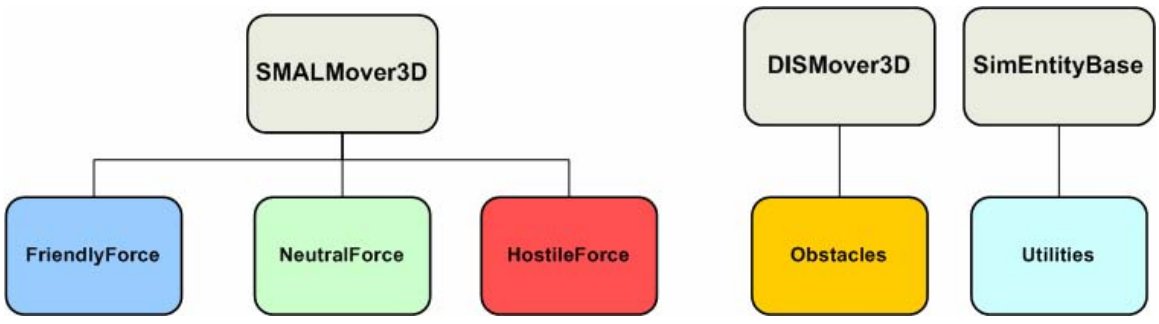


Figure 30. The behavior library inheritance structure used for this thesis.

As a general rule event graphs that were designed to represent intelligent agent behaviors and that required a structured, detailed parameter set (SMAL) were SMALMover3D (Friend/Hostile/Neutral). Obstacles, which in general have fewer parameter requirements, extend DISMover3D to ensure that they implement the Mover3D interface and can be recognized by other agents.

Utility objects, such as a Nautical Chart are not entities and have no real-world counterpart. Agents use the information contained in these objects but do not interact with them. These objects extend SimEntityBase so that the values that they contain are reset for each simulation run.

The majority of defined behaviors extend the Friendly, Neutral, and Hostile force types. These intermediate classes are used to establish affiliations between entities. This is accomplished by the creation of centralized lists for all entities that extend a given force type. This list is used for two purposes.

First it provides a means to identify entities of similar force type. As an example a military patrol craft can check to see if another object is part of the Friendly Force list. The result is either true or false. The military patrol craft does not have the ability to identify the specific force types of agents other than its own.

The second function of this list is to provide a means for directed communications within a force type. The radio communication object mentioned earlier allows the declaration of one or multiple recipients. When an entity desires to communicate with all similar entities on a radio circuit, this list is used to identify a defined group of intended recipients.

The final extension in the inheritance structure is the actual event graph model. This inheritance structure allows the contents of an event graph model to focus only on the specific events that are used to define a unique and distinguishable agent behavior definition.

C. IMPLEMENTING BEHAVIORS — EVENT GRAPH AUTHORING

The process of creating event graphs for agent behaviors is discussed in this section. This discussion is not intended to provide a 'how to use the interface' description of using Viskit but rather to examine all of the components used in terms of agent behavior modeling. Specific guidance on using the Viskit authoring tool can be found in the help and tutorial sections of the Viskit application.

The primary goal of creating an event graph behavior is to provide a complete behavior definition that can be used as is, expanded or modified. Accordingly it is imperative that users include detailed descriptions of every field in provided comment blocks. This ensures that future users of the library can understand the implementation.

1. Parameters

Parameters for the event graph are values that do not change during the simulation. They also represent values that must be provided when using this event graph in order for it to function properly. Table 3 lists the three primary entity extensions and their minimum parameter requirements.

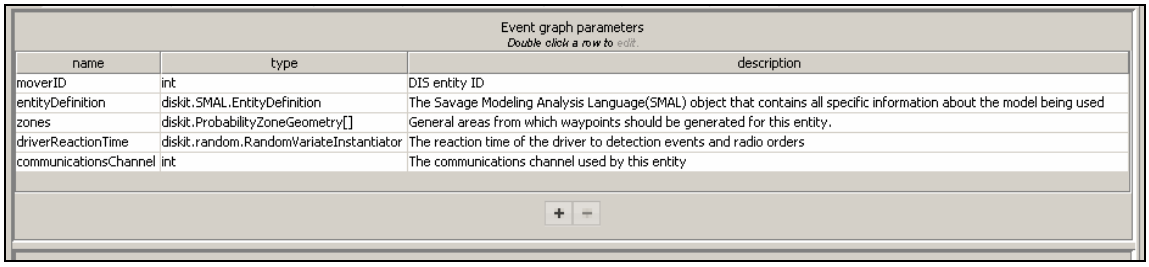
Entity Type	Minimum required parameters
SimEntityBase	• No required parameters
DISMover3D	• Mover ID# • Maximum Speed • Starting Position
SMALMover3D	• Mover ID# • SMAL Entity Definition

Table 3. The minimum required parameters for Viskit entities

SimEntityBase is the most basic type of entity and does not require any set parameters. Mover3D entities do have minimum parameter requirements defined by the classes that they extend.

Both DISMover3D and SMALMover3D must have a mover identification number. This unique number is used by the DIS protocol to distinguish between different entities and must be specified. DISMover3D also requires a maximum speed and starting position. These values are handled by the SMAL Entity Definition for SMALMover3D.

The super classes of the java inheritance structure require that the parameters identified in Table 3 must be included in the exact order listed. Additional parameters can be added to an event graph after the required parameters. For behavior modeling any values that will affect performance of the entity (e.g., sensor ranges, reaction times, and physical characteristics) need to be explicitly declared as a parameter. This ensures that individuals who use the behavior definition are fully aware of the specific values used when creating the simulation.


Event graph parameters

Double click a row to edit.

name	type	description
moverID	int	DIS entity ID
entityDefinition	diskit.SMAL.EntityDefinition	The Savage Modeling Analysis Language(SMAL) object that contains all specific information about the model being used
zones	diskit.ProbabilityZoneGeometry[]	General areas from which waypoints should be generated for this entity.
driverReactionTime	diskit.random.RandomVariateInstantiator	The reaction time of the driver to detection events and radio orders
communicationsChannel	int	The communications channel used by this entity

Figure 31. Depicts an example event graph parameter declaration for a SMALMover3D based event graph.

Figure 31 shows an example parameter set declaration for the Military Patrol Craft (MPC) event graph. In addition to the SMALMover3D parameter requirements, an array of zone geometry, a driver response-time variable, and a communication channel are listed. The inclusion of detailed descriptions ensures that users are fully aware of the purpose of each parameter.

2. State Variables

State variables, unlike parameters, are values or objects that change during the simulation. State variables can be any type of object. Figure 32 shows the MPC state variable set to illustrate this point.

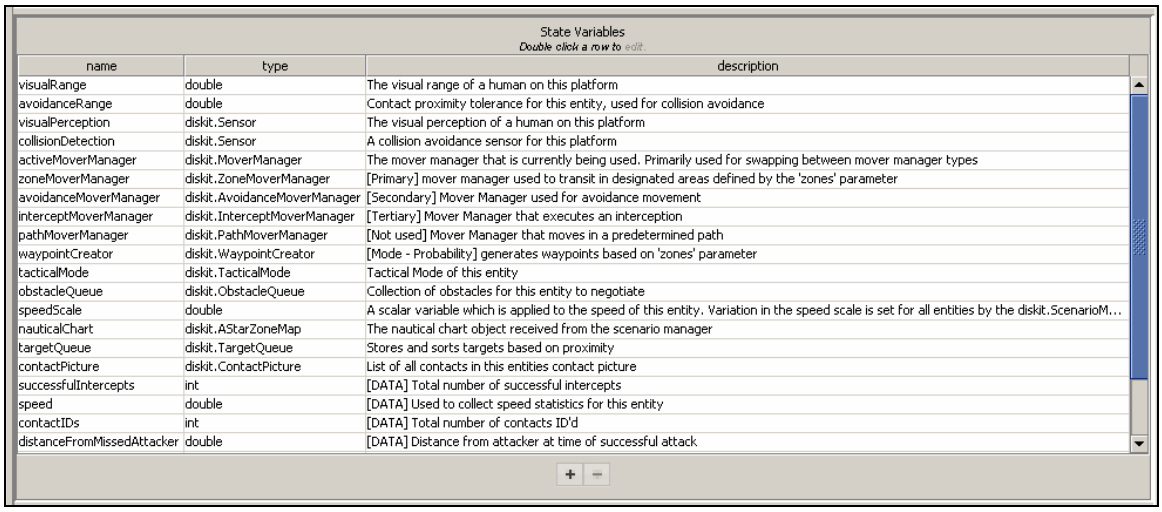


Figure 32. Example event graph state variables for a military patrol craft event graph.

The number of state variables can be very large. In order to maintain understandability and clarity a consistent pattern for listing state variables was used. While there is no formal requirement for ordering state variables having a consistent structure makes it much easier for users to compare event graphs and understand implementation differences. The pattern used in this project is listed in Table 4.

State Variable Type	**Description**
Sensor Ranges	The ranges of the sensors that are going to created.
Sensors	The actual sensor objects that will be created.
Mover Managers	All available mover managers. Includes a description of which mover manager is primary, secondary, etc. Mover managers that are not used are still included in case an author wants to use it later.
Waypoint Creator	The waypoint creator object that will be generating waypoints. The method of waypoint generation is explicitly stated for clarification.
Other Utility Objects	Any other objects or variables required throughout the event graph to implement the desired behavior.
Data Variables	Variables that are of interest for data collection and statistics. Explicitly marked for future reference.

Table 4. The organizational pattern for listing state variables in an event graph.

When state variable values change during a simulation run an opportunity exists to collect information about that state variable. This process is performed during the execution of an event and is discussed in the next section. The convention used for this project was to explicitly identify which state variables were included in the event graph model for the purpose of providing information to an analyst. While neither of these patterns are required for the simulation to run, a consistent naming pattern such as this is highly recommended for users planning to author a large number of event graphs.

3. Event Nodes

The event node in an event graph model is where state transition functions occur. The combination of events and their scheduling conditions form the graphical model representation in Viskit as shown in Figure 33.

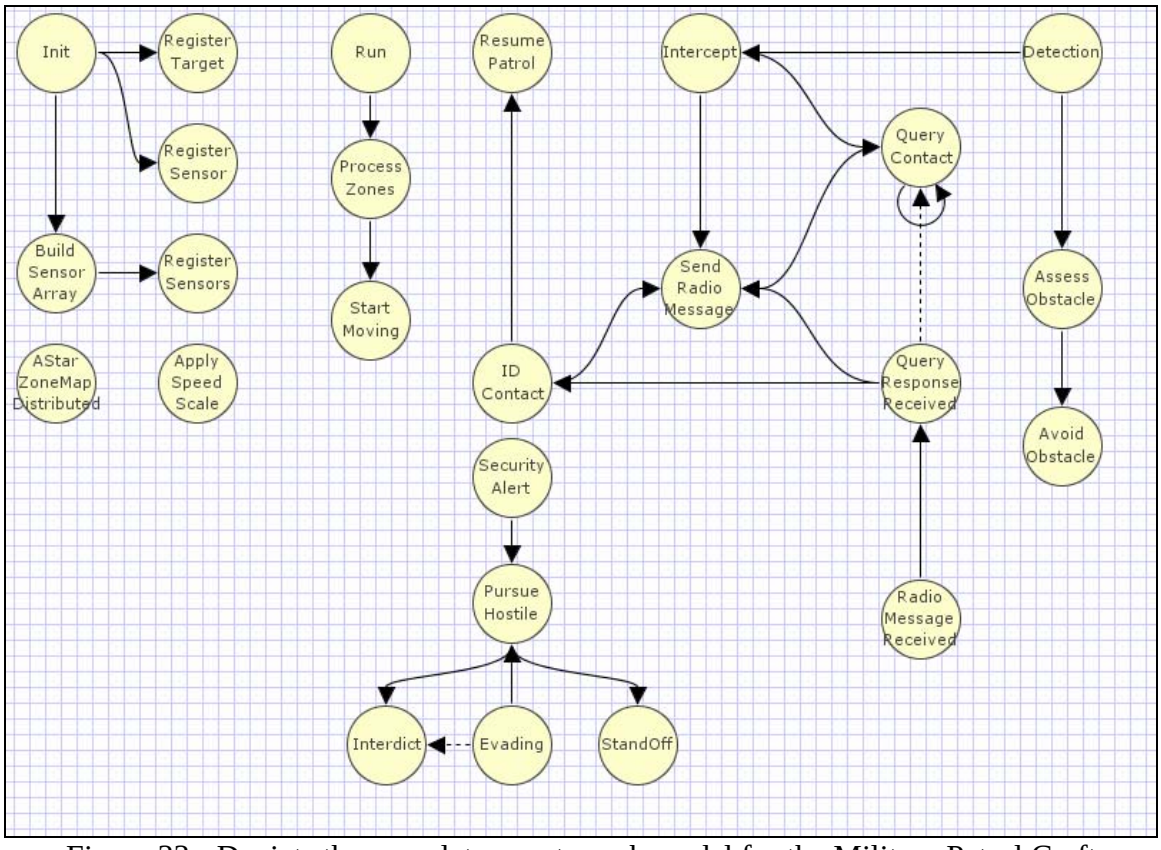


Figure 33. Depicts the complete event graph model for the Military Patrol Craft behavior definition.

The details for each event in Viskit are contained in the Event Inspector, which is accessed by clicking on the event. Figure 34 shows the Event Inspector for the Detection event in the MPC event graph. Each event has four major components: event arguments, local variables, a code block, and state transitions.

The arguments for an event indicate what types of objects must be included when this event is scheduled. Similar to a method call in Java, if a parameter is not of the right type or is omitted, the event will not be scheduled because its signature doesn't match.

Local variables are created and used within the event to perform some function. These variables are often used to define the conditions for scheduling edges between events but can be created for any purpose.

The code block section of this Viskit panel allows the entry of any additional Java code that does not meet the criteria of the other three event components. In this example the code is used to generate print statements for debugging purposes. The print statement in this line of code is commented out, but is nevertheless saved in case future debugging warrants the use of this message.

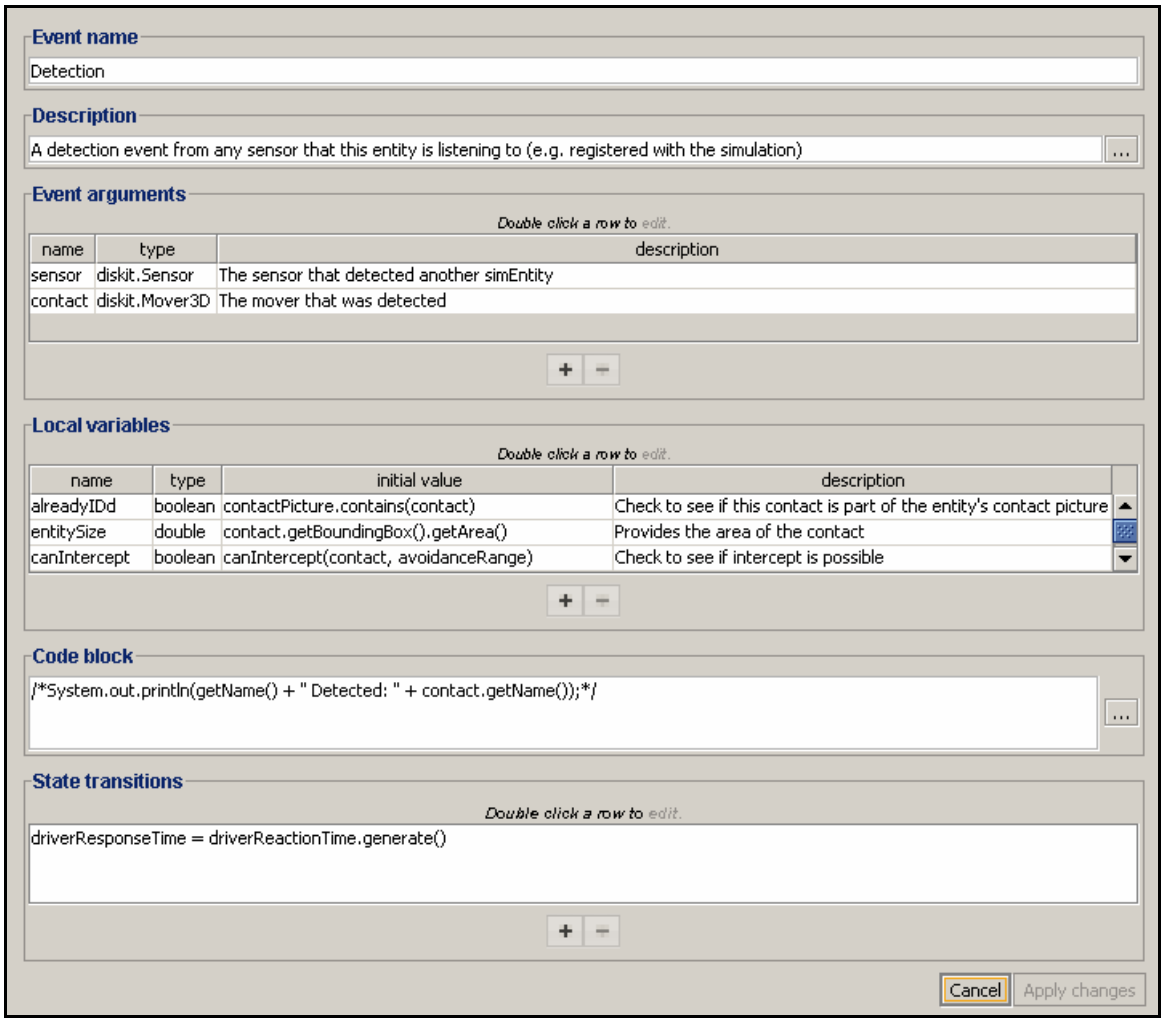


Figure 34. Event Inspector displaying the Detection event of the Military Patrol Craft event graph.

Finally, the state transitions block allows any state variable value to be changed by a specified function. In this example, the driver response time state variable value is being changed. This state transition represents an opportunity to record information about a variable. How this information recording occurs is discussed later in this chapter.

4. Scheduling Edges

Events are connected by scheduling edges. The Detection event has two scheduling edges. For this discussion we examine the scheduling edge that is designed to schedule an Intercept Event after a Detection Event occurs.

The Edge Inspector panel, shown in Figure 35, has three major components: a time delay, conditional expression, and edge parameters.

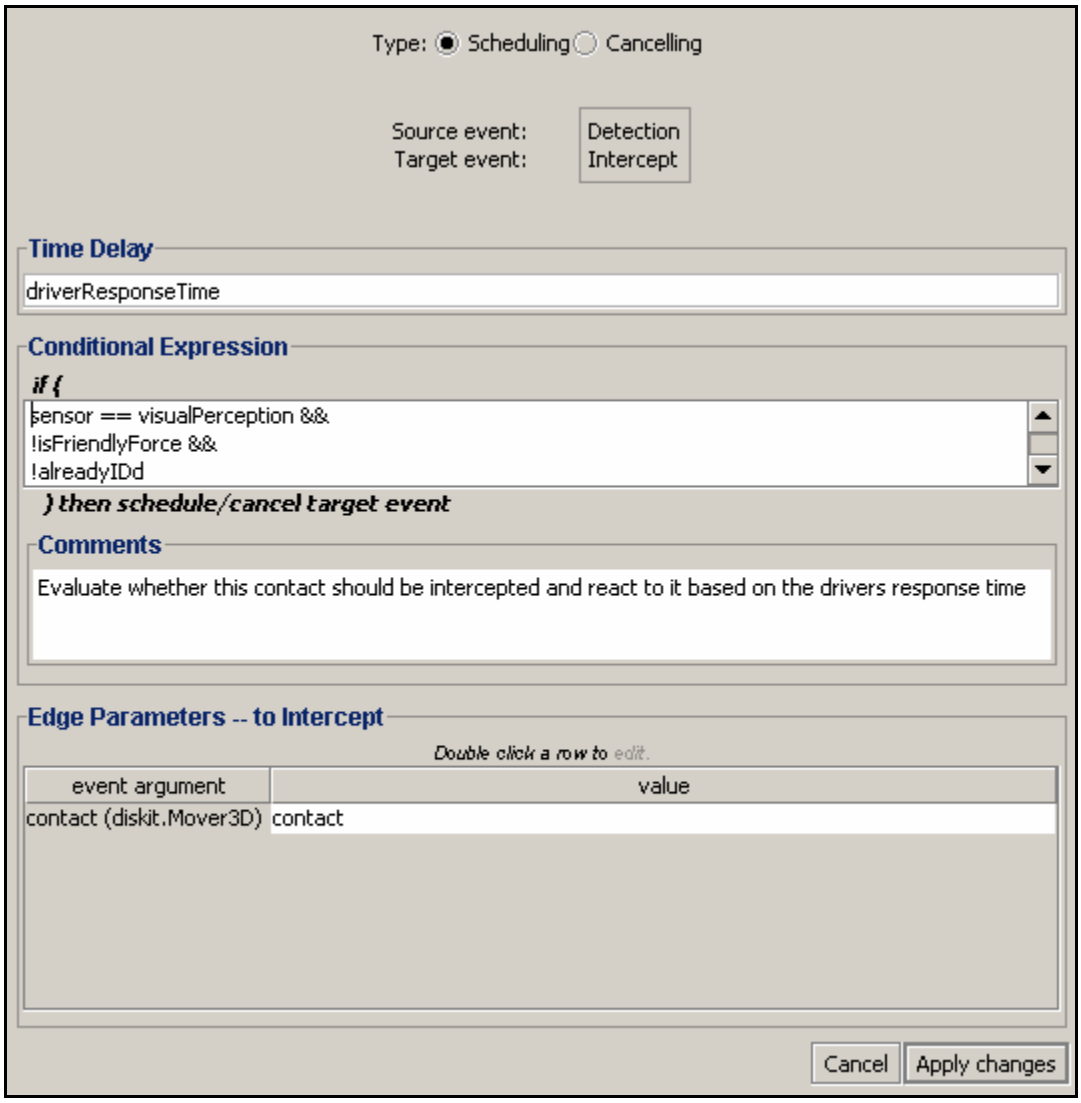


Figure 35. Scheduling edge inspector panel showing the details of the scheduling edge between the Detection and Intercept events of the military patrol craft.

The time delay indicates how far in the future to schedule the event or at what time to schedule the event on the event list. In this example the value that was created for driver response time is used for this purpose.

The conditional expression of a scheduling edge lists all of the conditions that must be met for the scheduling between events to occur. In Figure 35 the conditional expression is a check of which sensor detected a contact, whether or not it is on the friendly force list, and whether or not it has already been intercepted and identified. If these conditions are met then the detection event will schedule an intercept event.

Finally, each scheduling edge can pass parameters between events. The required parameters are defined by the receiving event that is being scheduled. In this case the Intercept event requires a Mover3D object which it calls 'contact'. The Intercept event needs to have the contact so it can perform the intercept calculations for that contact. In this example, the detected contact of the Detection Event is passed across the edge to the Intercept Event once the scheduled event occurs.

5. Canceling Edges

Canceling Edges are used to cancel or remove scheduled events on the event list. In graphical form this edge is represented by a dotted line. Figure 36 shows the canceling edge that exists between the Query Response Received event and the Query Contact Event excerpted from the MPC event graph in Figure 33.

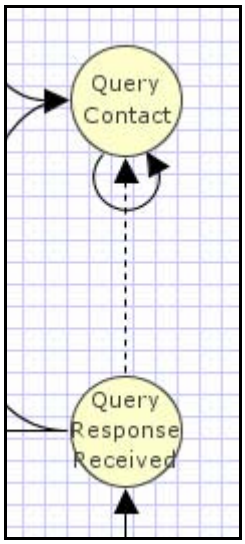


Figure 36. Canceling edge between the Query Response Received and Query Contact events of the military patrol craft event graph. (Excerpted from Figure 33)

Canceling edges do not have a time delay. The event that is being cancelled is removed from the event list immediately. The execution of a canceling edge is still dependent on a conditional expression and contains edge parameters. In this example a Query Contact event occurs and repeats until a response is received. If a response is not received then the MPC will continue to query the contact. Once a query response is received, the Query Response Received event cancels the next Query Contact event on

the event list. In this case the condition required for such a cancellation to occur is that the responding contact no longer needs to be scheduled for a query.

6. Tactical Behavior Modeling

The final consideration for agent design was how to implement tactical behaviors for this project using the structure that has been defined in this chapter. The three types of 'intelligent' agents in this simulation are Friendly, Hostile, and Neutral. For our purposes 'Friendly' refers to U.S. or coalition-partner military assets and personnel. Hostile refers to those entities that intend to do harm or damage to Friendly agents or the harbor environment. Finally, Neutral agents are disinterested agents who are conducting their normal operations or behaviors (e.g., a fishing boat).

The approach taken for friendly agent design included four major requirements:

- The ability to define and use a layered defense strategy.
- The ability to define and use patrol areas and areas of responsibility.
- The ability to use a simulated Command and Control (C2) system.
- The ability to coordinate efforts and actions.

A layered defense strategy is a common approach for units in a defensive posture. The original AT/FP tool (Harney 2003) provided the user with the ability to define range circles for an identification zone, interception zone, and engagement zone. The original implementation of this concept is shown in Figure 37.

Figure 37. Depicts the use of a zone defense system in the original AT/FP simulation tool (from Harney 2003)

This concept was also used in this simulation but modified so that it could easily map to the Simkit/Diskit implementation. Zones in this framework are represented by a Sphere Cutter Sensor. Entities use the Detection and Un-Detection events from these sensors to determine when a contact has entered or exited a zone. Figure 38 shows the new implementation of these zone objects.

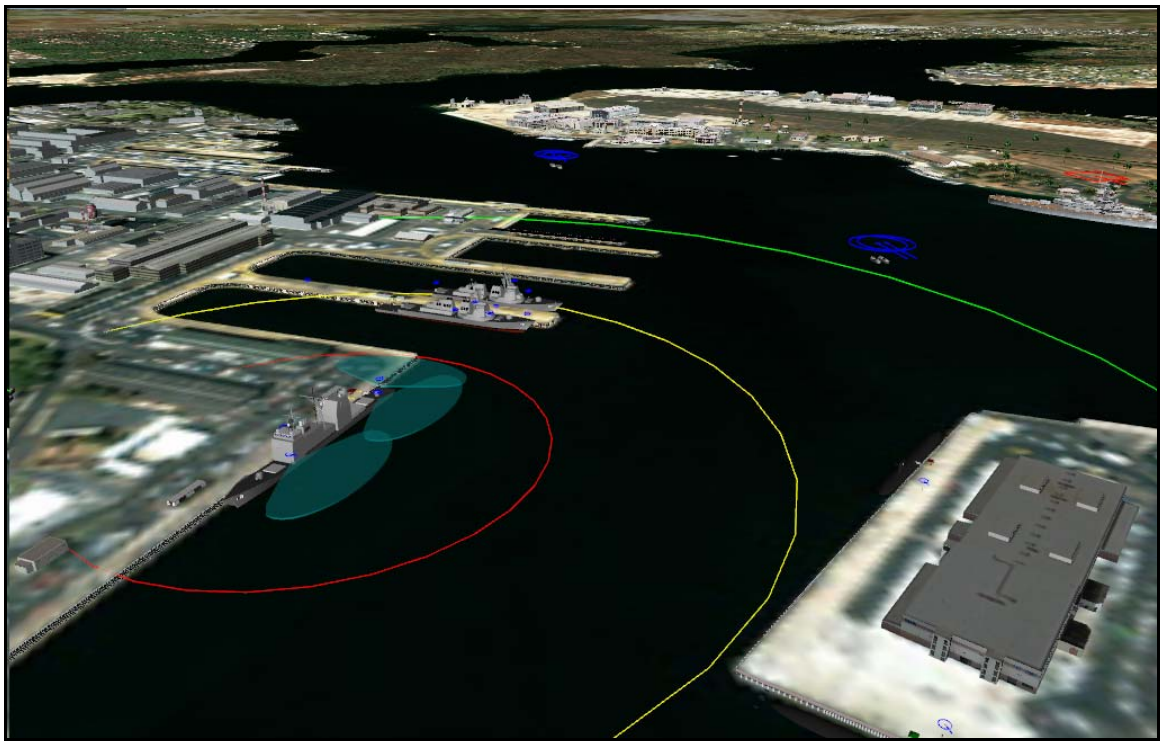


Figure 38. Depicts the implementation of zones in the new version of the AT/FP simulation tool.

Patrol zones and areas of responsibility were achieved by utilizing zone geometry objects and assigning agents to them. This implementation loosely defined where these agents were expected to be during the course of their duties.

The creation of radio communication objects allowed for a C2 implementation at the event graph level. Figure 39 shows the behavior event graph for a Ship Self Defense Force (SSDF) person. The job of this individual is to man a weapon on the outside of the ship and when directed to fire at a contact that has been identified as hostile. This person is not at the station until there is a threat in the area and the ship has detected and announced a threat over a communications circuit. Once the person mans the weapon they do not shoot until they have been given a “batteries release” command over a communications circuit.

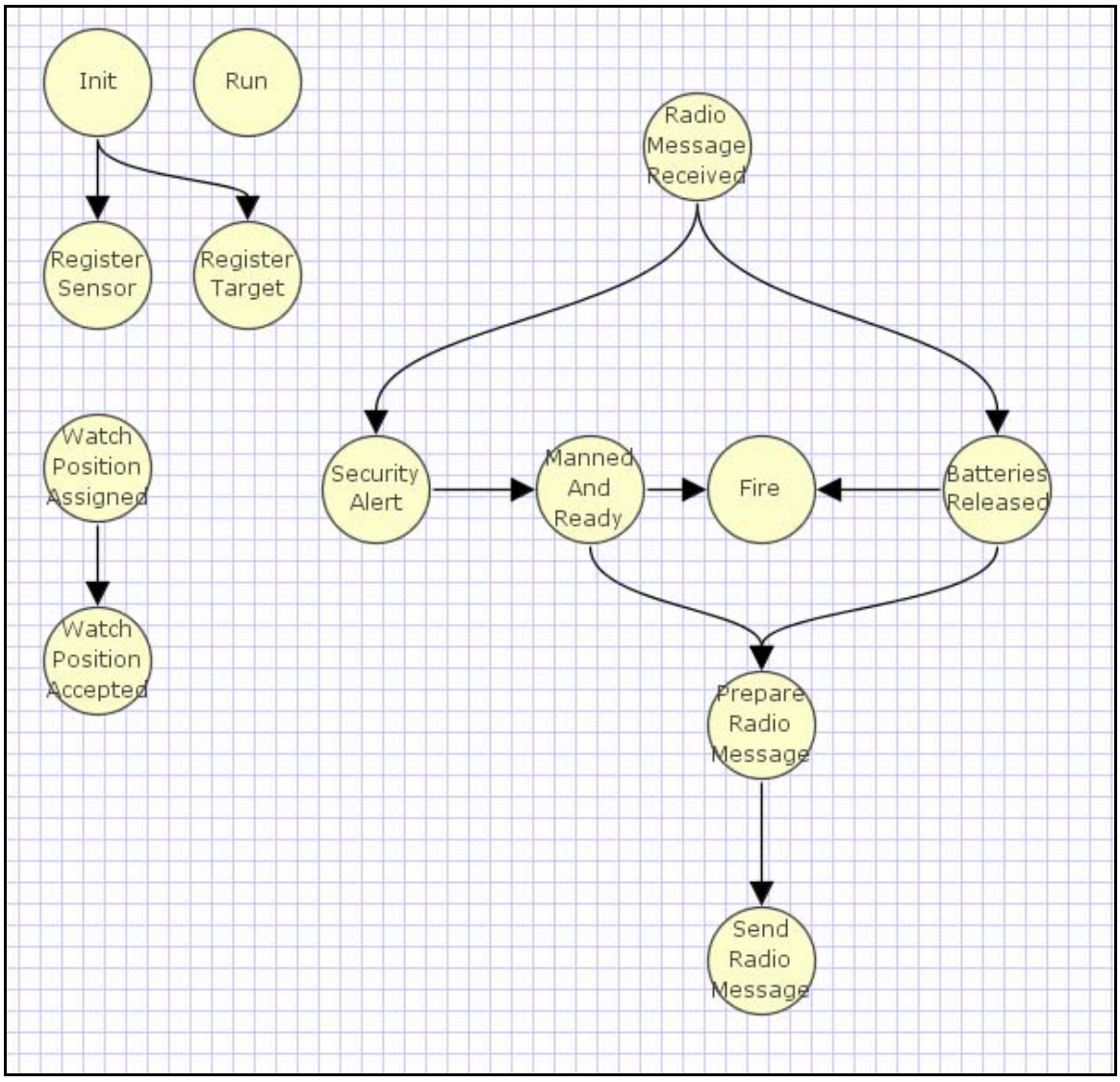


Figure 39. Event graph of a Ship Self Defense Force agent. Depicts a behavior definition that is completely dependent on functional command and control.

This individual is completely dependent on the communications in the real-world. Accordingly the behavior definition is completely dependent on the scheduling of a Radio Message Received event. Following this simple pattern it was possible to model a command and control structure.

The implementation of Hostile agents was broken down into two event graphs: a Terrorist Cell Planner event graph and an Explosive Laden Vessel event graph.

The Terrorist Cell Planner (TCP), shown in Figure 40, was designed to develop and order attack plans for the simulation. To create attack plans the TCP follows a

simple process that begins when it receives a copy of the Nautical Chart. This occurs in the AStarZoneMapDistributed event.

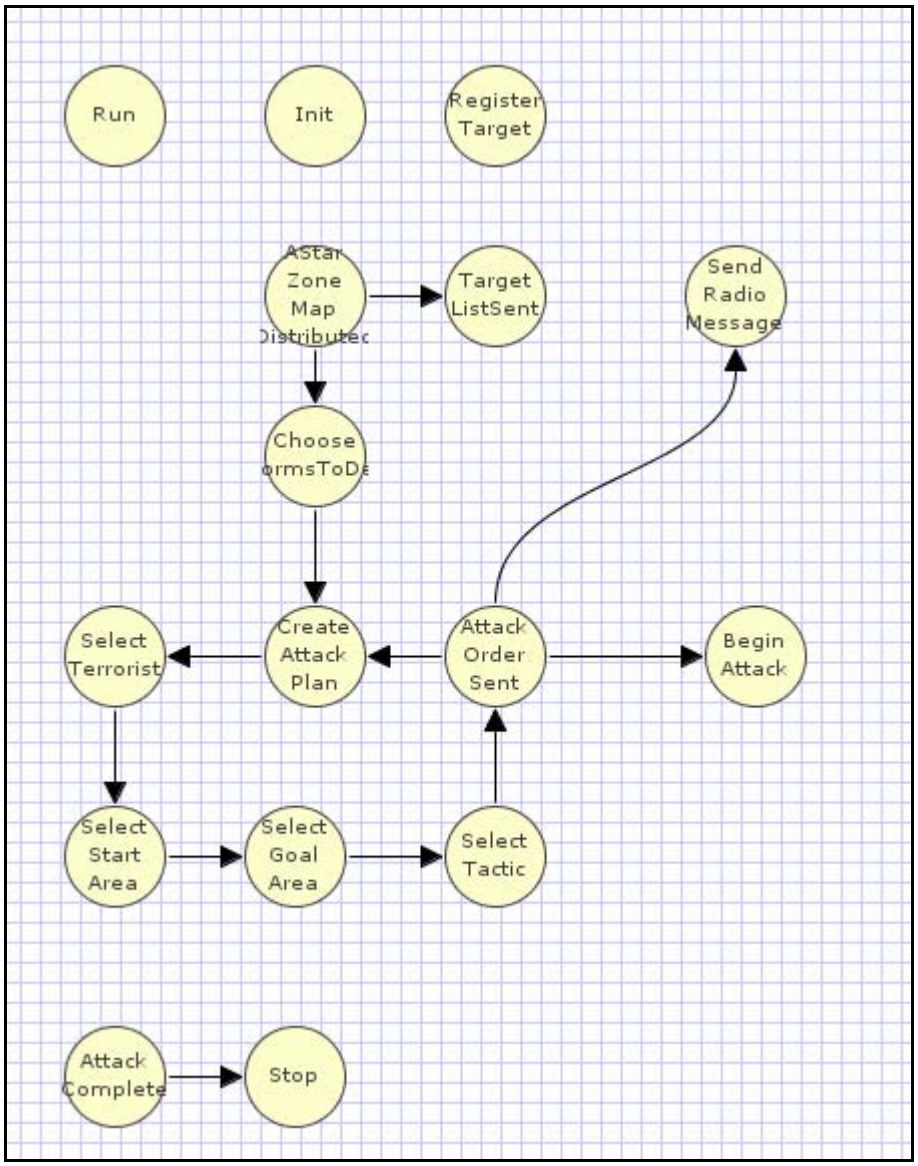


Figure 40. Depicts the terrorist cell planner event graph

After receiving a copy of the Nautical Chart the TCP provides all Explosive Laden Vessels (ELV) with a list of target types sorted by priority. The priority of the target is determined by its place on the list with the first target having the highest priority. The TCP then creates an attack plan by selecting a specific terrorist, randomly picking a perimeter zone from the Nautical Chart as a start location, and designating the water zone

on the map that is the closest to the targets of interest. Finally, a tactic is identified. The combination of these components makes up an attack plan. Each selected ELV receives an attack plan for execution.

The ELV was designed to receive and execute the attack plans that it receives from the TCP. The ELV event graph is shown in Figure 41.

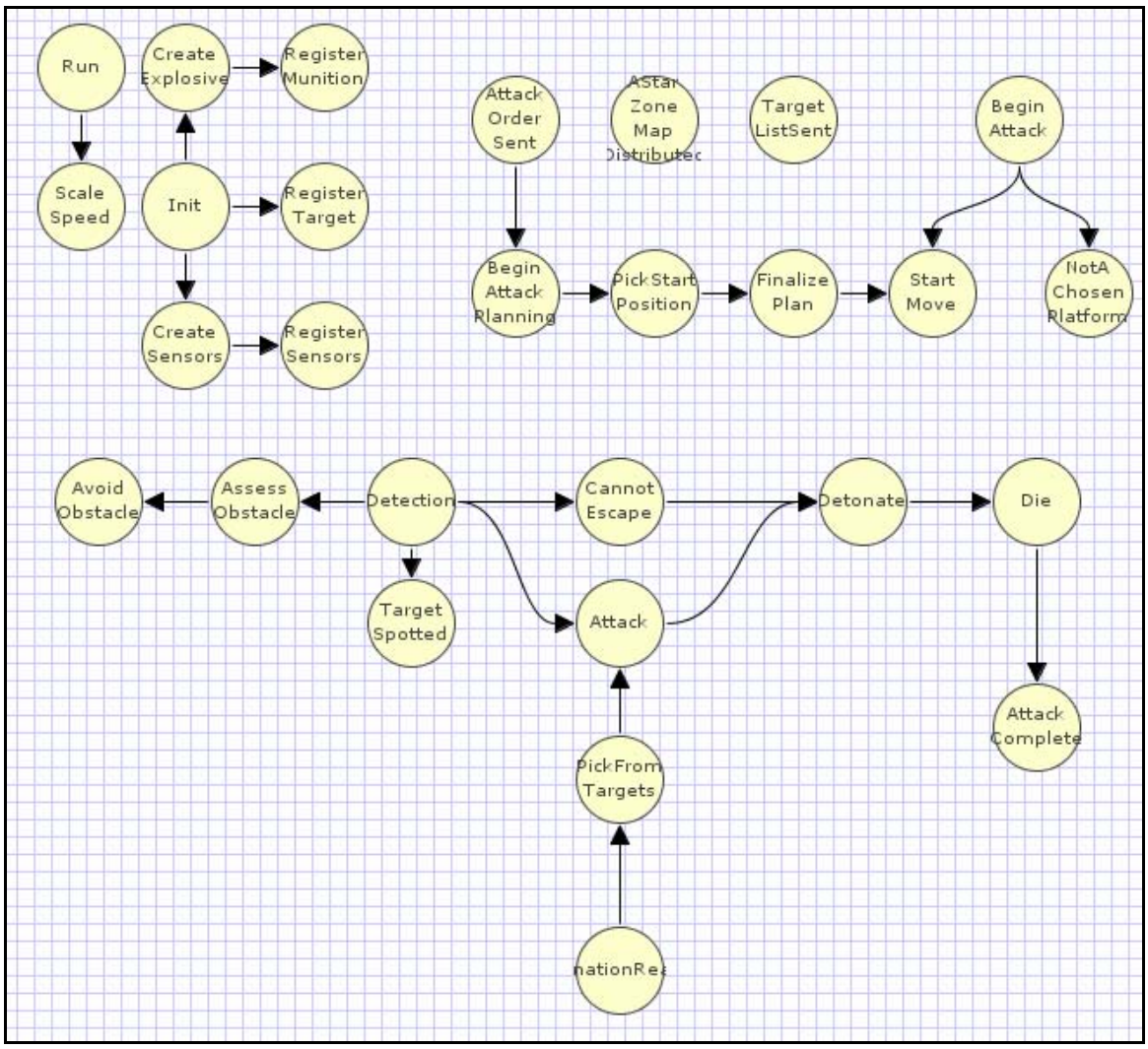


Figure 41. Event graph for an Explosive Laden Vessel

The events Target List Sent and Attack Order Sent in the ELV event graph are scheduled by the TCP. This is accomplished through the use of SimEventListener connections which are discussed in the next section. After receiving the attack plan the terrorist uses the A* search algorithm of Diskit to plan a path from the ordered starting

zone to the destination zone. After the terrorist begins its attack it will avoid obstacles and attempt to attack the primary target type. If the target type is spotted the ELV will attack and attempt to blow it up. At the conclusion of an attack the results are passed back to the TCP via the Attack Complete event which is in both the ELV and TCP event graphs.

The event graph implementations shown in this chapter are examples of how complex behavior definitions can be created using Viskit. There is no limit to the types of behaviors that can be modeled with the event graph methodology. The true benefit of this tool is in the automatic code generation. The generated source code for the MPC is in excess of five hundred lines of code and the ELV source code is just under nine hundred lines of code. Viskit has provided a means to create large executable event graph based computer models in a few hours that before would have taken weeks or months to code by hand. This is a significant accomplishment.

D. CREATING A SIMULATION — ASSEMBLY AUTHORING

A user constructs a scenario using the Assembly Panel. This panel allows users to select which behaviors they want to use and to connect them together to form a simulation. Figure 42 shows the assembly panel with a scenario for Bremerton, Washington. The Assembly is something that does not exist in Simkit. In Simkit a programmer would create a main execution class and piece together a simulation within the body of that class. In Viskit the assembly performs the same function but provides a visual representation of the construction of the simulation.

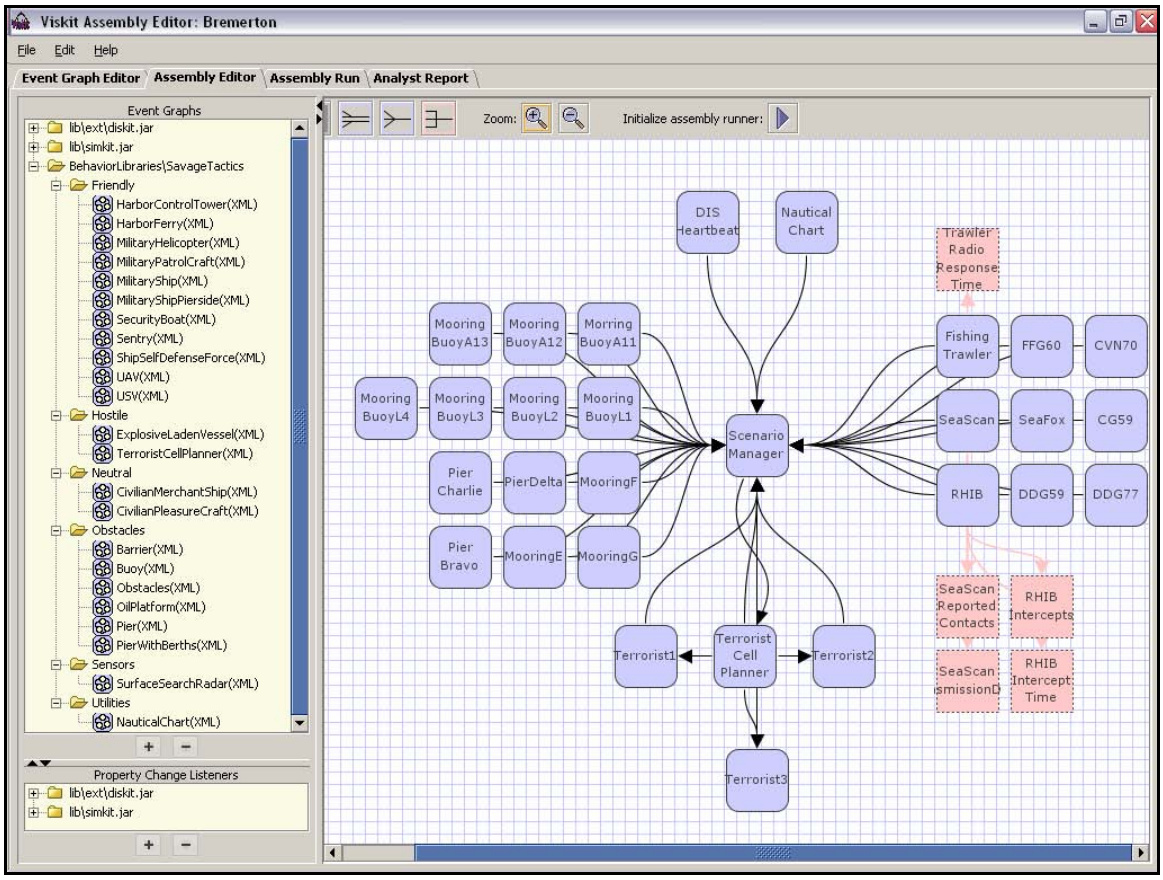


Figure 42. The Viskit assembly panel with a Bremerton, Washington scenario loaded.

1. Behavior Libraries

After event graph models of all of the behaviors or objects for a simulation have been created the user can select from a library of the event graphs to include in the simulation. The complete set of Behavior Libraries for this project is shown in Figure 43.

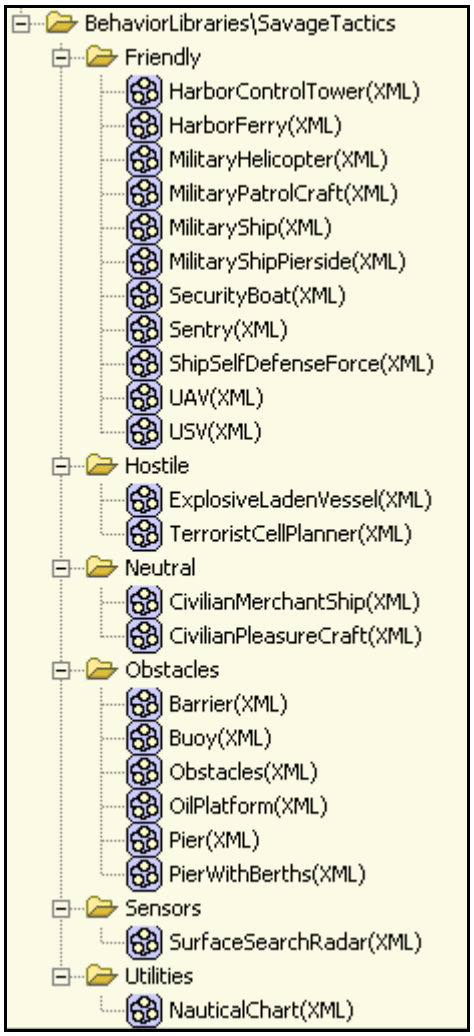


Figure 43. Display of the complete Behavior Library for this project.

Users simply select a behavior from this library and drag-and-drop it onto the assembly panel. This creates an instance of the behavior and a SimEntity Node (the purple box objects in Figure 42).

2. SimEntity Nodes

The SimEntity Node is used to enter the parameter values that are identified in an event graph. SimEntity Nodes also allow the simulation author to provide detailed information and labels about what they believe an entity represents. In this example Military Ship event graphs are labeled with hull identification numbers, and piers and obstacles are labeled with the names of the objects that they represent.

3. Parameter Entry

The true value of implementing SMAL is realized in terms of parameter entry for a SimEntity. The Military Patrol Craft (MPC) event graph listed a SMAL Entity Definition as a parameter. When this is expanded in the assembly panel all of the parameters of the Entity Definition are accessible. Figure 44 shows the expansion of the Entity Definition to the Dynamic Response Constraints for the Rigid Hull Inflatable Boat (RHIB) in the Bremerton scenario.

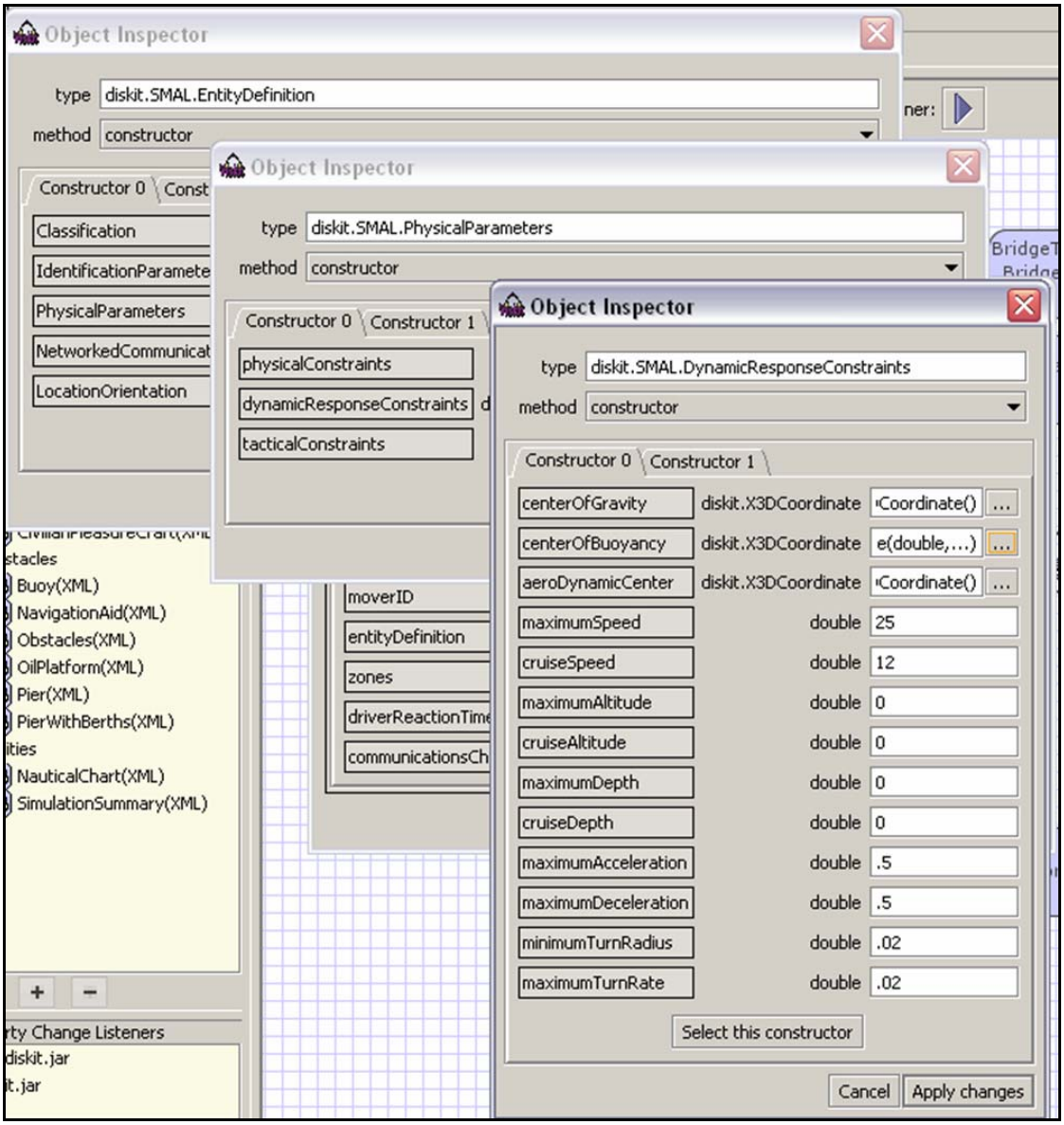


Figure 44. SMAL Dynamic Response Constraints exposed for the Rigid Hull Inflatable Boat (RHIB) in the Bremerton, Washington scenario.

SMAL is not the only feature-rich parameter set however. One of the many features of Simkit is a robust random number generation capability. Recall that the event graph for the Military Patrol Craft had a parameter called Driver Reaction Time. This parameter uses a the Random Number Factory of Simkit to generate random numbers based on a distribution. Figure 45 shows the Driver Reaction Time field fully expanded for user input.

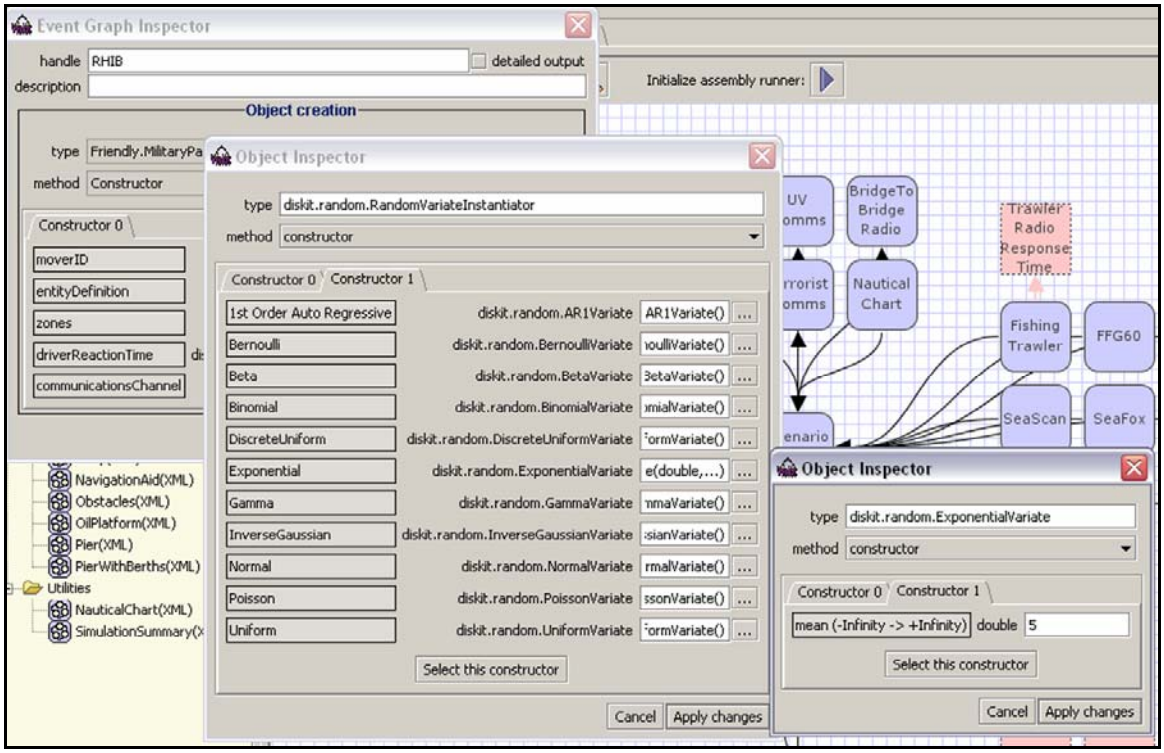


Figure 45. Example of using Simkit's Random Number Factory to create random numbers based on a distribution.

In this example the reaction time is identified as having an Exponential distribution with a mean of five. When the driverReactionTime.generate() method call is performed in the Detection event discussed earlier, the number returned will be based on that distribution.

4. SimEventListener Connectors

The lines that connect various nodes to each other in the assembly panel are SimEventListener connections. Earlier we showed that the Terrorist Cell Planner (TCP) and the Explosive Laden Vessel (ELV) had a set of identical events. Additionally, we said that these events provided a means for the two agents to share information. This is accomplished via a SimEventListener Connections. In the Bremerton example the TCP and all three terrorist SimEntity nodes are connected with SimEventListener connections.

5. Property Change Listener Nodes

In our discussion of state variables it was noted that state transitions provide an opportunity for data collection. This is accomplished through the use of Property Change Listener (PCL) nodes. In the Bremerton example the pink boxes represent Property Change Listeners. Each PCL is listening for changes to a specific state variable. Figure 46 shows the PCL for the RHIB Intercept Time property.

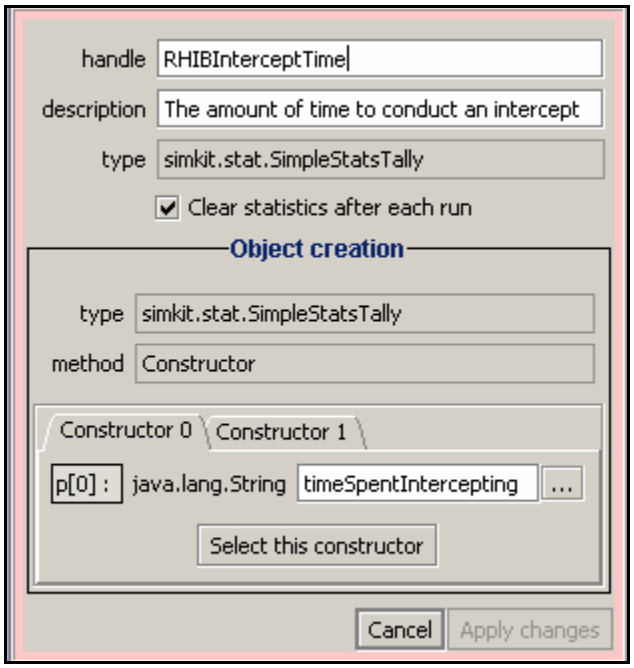


Figure 46. Property Change listener for RHIB Intercept Time from the Bremerton scenario.

In this panel the user has specified that he is interested in collecting information on the amount of time it takes the RHIB to conduct an intercept. This node is connected

to the entity that it is listening to via a PCL connection. By opening that connection the specifics of the listener are revealed as shown in Figure 47.

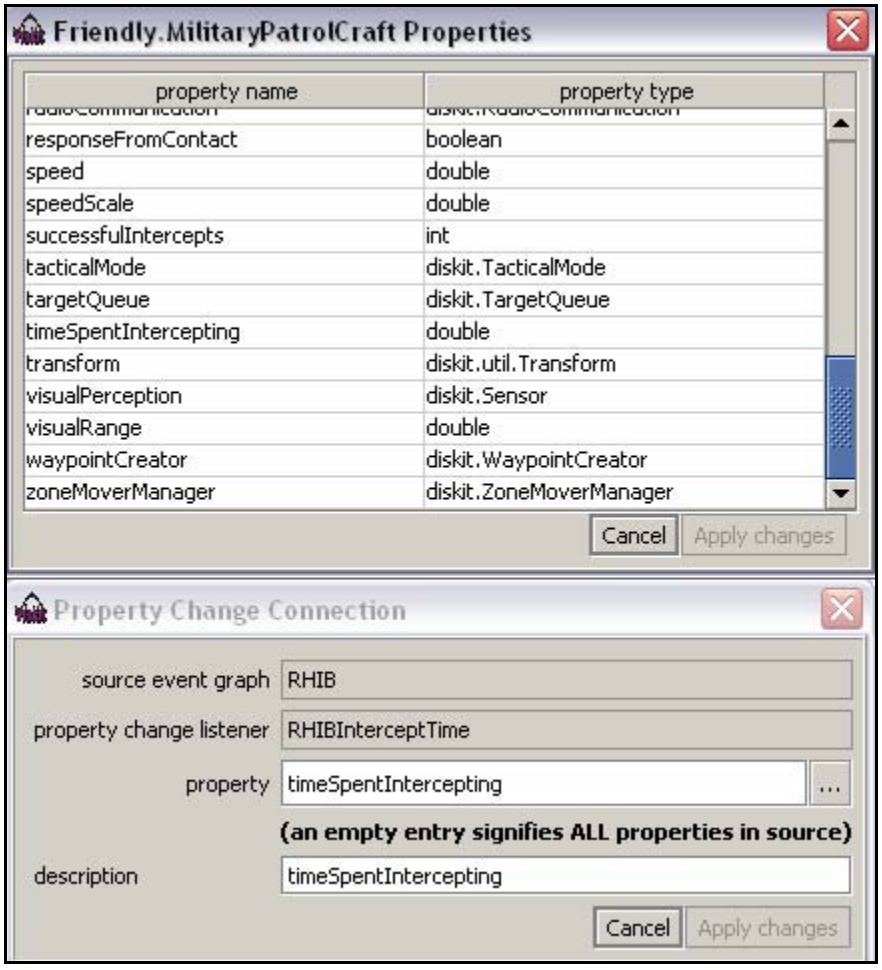


Figure 47. Depicts the listener connection details for the time spent intercepting state variable of the Military Patrol Craft.

Figure 47 shows that all of the state variables are available for data collection. In the event graph for the Military Patrol Craft when an Intercept is going to be conducted part of the calculation is the amount of time that it is going to take to execute the intercept. Every time this event occurs the calculation is performed and the 'timeSpentIntercepting' state variable is updated as shown in Figure 48.

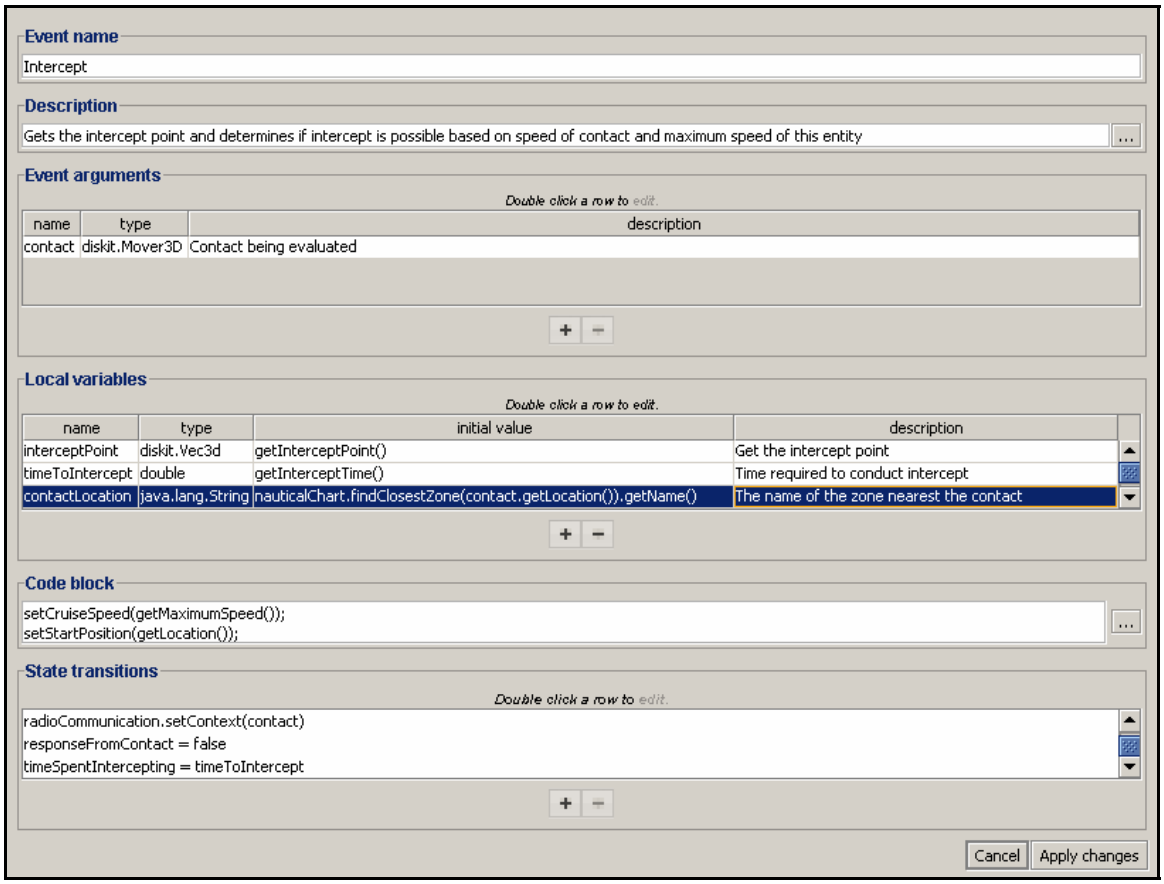


Figure 48. Shows the state variable transition function for timeSpentIntercepting in the Military Patrol Craft event graph.

This change in value is recorded by the PCL. At the end of each replication statistics are calculated for the total count, mean, standard deviation, and variance for the values of this property. Additionally, at the end of the entire simulation summary statistics are provided for the data collected.

6. Scenario Manager / DIS Heartbeat

Two final components of the assembly are critical to the simulation and are worth mentioning. The Scenario Manager is the centralized point of the simulation and as its name suggests is responsible for managing the simulation. All movers and objects should have a SimEventListener connection with the Scenario Manager. If the connection does not exist then that SimEntity is not registered with the simulation and does not participate. The Scenario Manager is also where the information necessary to create a DIS connection is entered. For the purposes of this discussion it is sufficient to note that a DIS simulation with 3D movers can not run without a Scenario Manager.

The other component is the DIS Heartbeat. This node is required if the simulation is driving a graphical display using DIS. The node provides the interval within which information must be sent over the internet. When this node is set to run, the simulation runs in real time since it is driving a real time display. When this node is not active, the simulation runs in discrete event mode and runs much faster by leveraging the event list methodology discussed earlier.

E. SIMULATION RESULTS — ANALYST REPORT GENERATION

One of the most powerful new features of Viskit is the ability to generate a report from a simulation. Early in the project this functionality was identified as a critical requirement to facilitate use of the simulation for real-world harbor security analysis. In addition to report generation, the ability for an analyst to annotate the report both before and after a simulation was desired. Enhancements to Viskit provide this capability allowing the user to author and generate a detailed report before, during, and after the design and execution of a simulation.

1. Analyst Report XML Document

Since Viskit provides the ability to create large-scale simulations the potential information for any report could be quite large. Accordingly, it was important to define a structure for capturing and storing the information that could be used to populate an analyst report. Table 5 lists the sections for the analyst report and provides a brief description of their purpose.

Report Section	Description
Executive Summary	Provides a summary of the simulation design, execution and findings.
Simulation Location	Provides a description of the environment for the simulation. Also provides the ability to include to images of the location being modeled.
Simulation Configuration	Autogenerated image of the Viskit assembly panel. Includes detailed parameter values for all entities in the simulation.
Behavior Definitions	Autogenerated images of the behavior event graphs that were used. Includes a detailed table of all parameters and state variables in a simulation.
Statistical Results	Provides a report of the replications and summary statistics for all of the property change listeners in the assembly. Also includes the ability to display charts of the simulation statistics.
Conclusions/Recommendations	Analyst describes conclusions and recommendations from the simulation and can make recommendations for future work.

Table 5. Listing of the sections of the Viskit generated analyst report

With the exception of analyst comments, environment images, and statistics, all of the information required for this report is contained in XML form in Viskit. JDOM was used in the Java classes of Viskit to use these various XML documents to create the analyst report XML document described above. The analyst report document serves as the local reference to a specific simulation. All of the information required to construct a simulation is added to this document and the entire report document is date and time stamped. The analyst report document is the sole resource used to create a final generated report.

2. Report Statistics XML Document

Statistics results from a replication run are not normally saved in XML form. The default Viskit output for statistics is text reports for each replication and a final summary

report at the end of a simulation run. Figure 49 shows an example of the text output from a run of the Bremerton, Washington, scenario.

```
Output Report for Replication #5
reportedContacts[0]     2       7.0000  8.0000  7.5000  0.5000  0.7071
transmitDelay[1]        3       4.6721  5.0826  4.8647  0.0426  0.2064
successfulIntercepts[2] 1       7.0000  7.0000  7.0000  0.0000  0.0000
timeSpentIntercepting[3]        2       0.0000  16.2452 8.1226  131.9531        11.4871
radioResponseDelay[4]   0       ?       -?      0.0000  0.0000  0.0000

Summary Output Report:
reportedContacts.mean (TALLY)
5  1.0000  7.5000  3.9000  6.9250  2.6315
transmitDelay.mean (TALLY)
5  4.6084  5.2980  4.9936  0.0722  0.2687
successfulIntercepts.mean (TALLY)
5  0.0000  7.0000  3.3000  7.7000  2.7749
timeSpentIntercepting.mean (TALLY)
5  0.0000  30.7293  13.4955  146.5039  12.1039
radioResponseDelay.mean (TALLY)
5  0.0000  40.0396  15.5757  455.4863  21.3421

******************************
******************************
```

Figure 49. Displays the normal text output for statistics information for Viskit.

This example shows that the information provided for console output is not very descriptive and would be of limited use without significant annotation. Figure 50 shows the implementation that was used to solve this structure problem. Since the analyst report and all of its supporting data sources are in XML format it was logical to transform the statistics output of Viskit into XML. This format sorts the statistics data by entity. Each entity can have multiple data points for statistical analysis. For each data point there is a replication report which lists the statistical output for each replication of the simulation. Each entity also has a summary report which lists the summary statistics output for all of the entity's data points. This structure organizes the statistical output of Viskit into a format that is much easier to incorporate into the analyst report document.

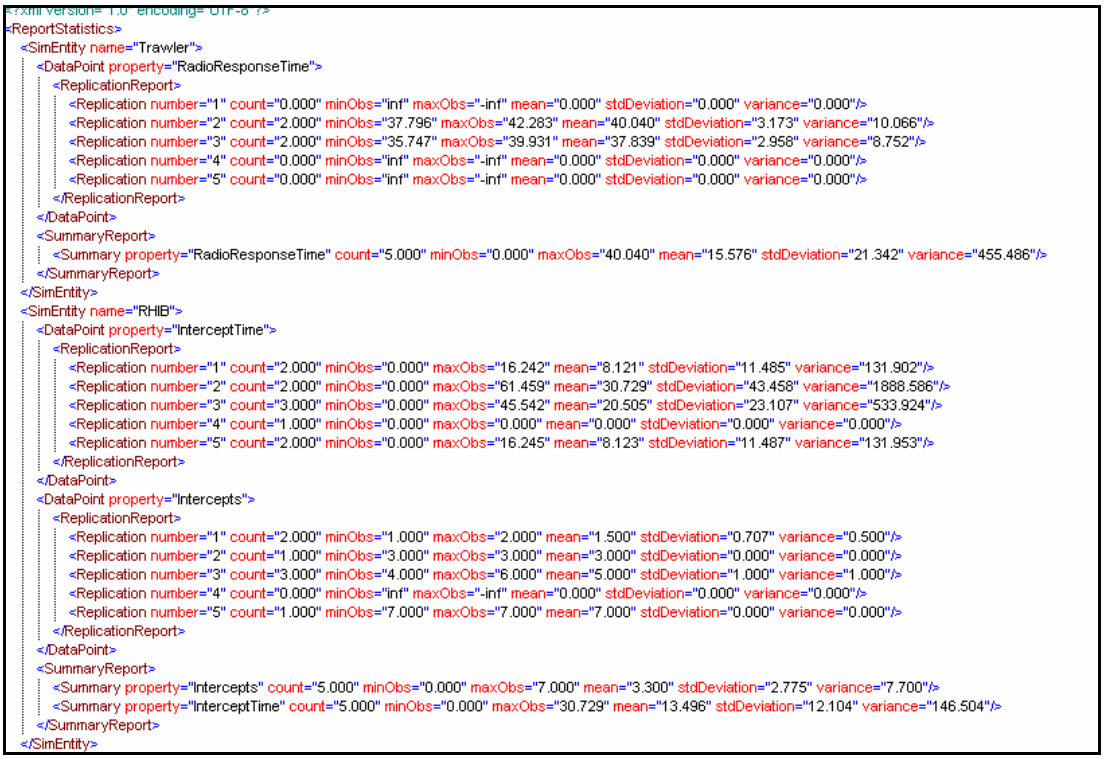


Figure 50. Example of sorted statistics in XML format from a Viskit simulation run

3. Report Generation

Once a finalized analyst report document is created it is necessary to transform it into a usable report format. For this project the data contained in the analyst report XML document is transformed into an HTML document by applying an XSLT designed for that purpose. The following example shows how the Simulation Location portion of the analyst report is generated with a detailed look at the underlying components.

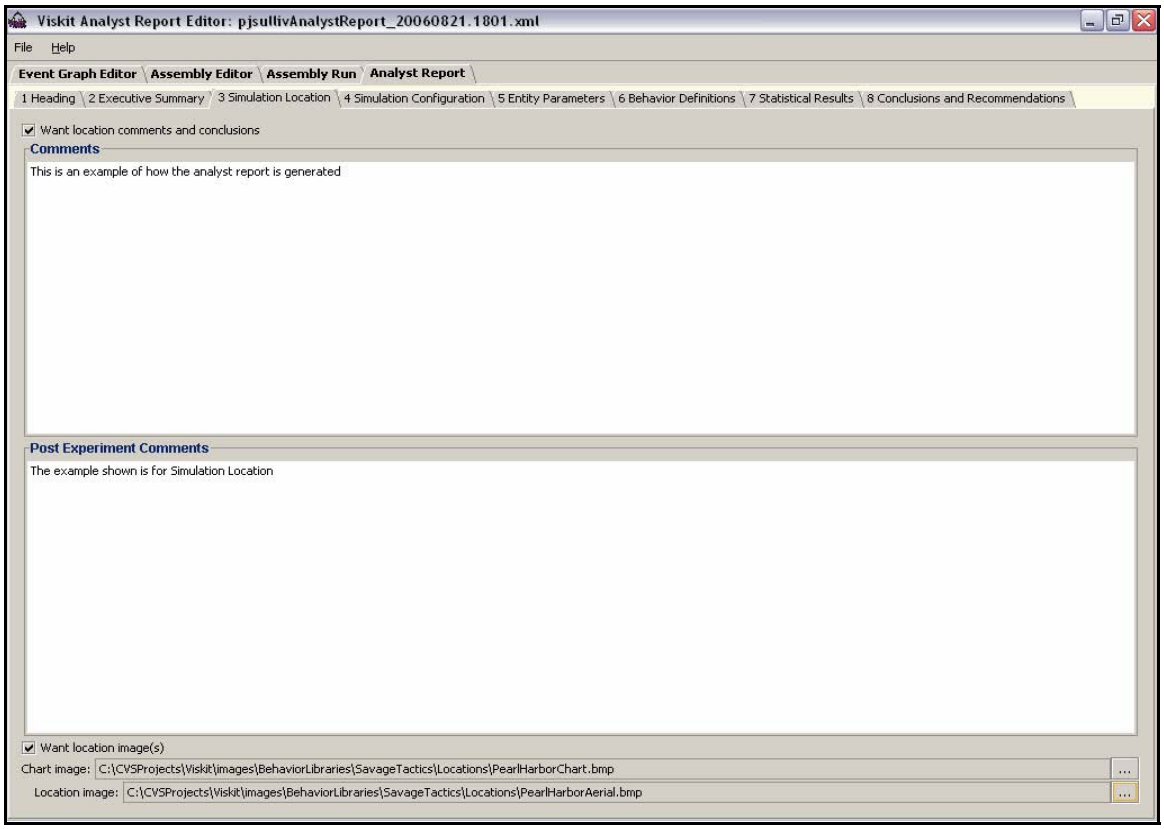


Figure 51. Analyst report panel for the simulation location portion of an analyst report.

Figure 51 shows the user interface for the Simulation Location portion of the analyst report. In this example the user has selected the boxes for comment and image generation indicating that they would like both comments and images included in this part of the report. Additionally, comments have been typed in both fields and two image file locations have been provided. When the analyst report is saved in Viskit this information is stored in the Analyst Report XML document as seen in Figure 52.

```
<SimulationLocation comments="true" images="true">

   <SLComments text="This is an example of how the analyst report is generated" />

   <SLConclusions text="The example shown is for Simulation Location" />

   <LocationImage
        dir="C:\CVSProjects\Viskit\images\BehaviorLibraries\SavageTactics\Locations\
             PearlHarborChart.bmp" />
   <LocationImage
        dir="C:\CVSProjects\Viskit\images\BehaviorLibraries\SavageTactics\Locations\
               PearlHarborAerial.bmp" />
</SimulationLocation>
```

Figure 52. Analyst report XML for the simulation location portion of the analyst report

The first line of the XML labeled ‘SimulationLocation’ indicates that this is the Simulation Location section of the report and that both comments and images are desired. The next line, labeled ‘SLComments’ identifies the field as the comments entry and includes the text that was entered in Viskit. The field labeled ‘SLConclusions’ provides the same information but for the conclusions section. Finally, the provided location image information is located in the LocationImage fields. The final step in the process is to transform the XML representation of this section of the report into a viewable, formatted HTML document.

```
<!--Simulation Location templates-->
<xsl:template match="SLComments">
<p align="left"><b>Simulation Location</b></p>
<p align="left"><i><u>Analyst Considerations</u>:</i><font color="#00006C"><xsl:value-of
select="@text" /></font></p>
</xsl:template>

<xsl:template match="SLConclusions">
<p align="left"><i><u>Post-Experiment Analysis</u>:</i><font color="#00006C"><xsl:value-
of select="@text" /></font></p>
</xsl:template>

<xsl:template match="LocationImage">
<p align="center">
<xsl:element name="img">
<xsl:attribute name="border"><xsl:text>1</xsl:text></xsl:attribute>
<xsl:attribute name="src">file:///<xsl:value-of select="@dir" /></xsl:attribute>
<xsl:attribute name="width"><xsl:text>640</xsl:text></xsl:attribute>
<xsl:attribute name="height"><xsl:text>480</xsl:text></xsl:attribute>
</xsl:element>
</p>
</xsl:template>
```

Figure 53. XSLT used to convert the simulation location portion of the analyst report from XML to XHTML.

This transform occurs when Viskit applies an XSLT Stylesheet to the analyst report XML. Figure 53 shows the portion of the Stylesheet that applies to the Simulation Location portion of the Analyst Report. In general terms the XSLT Stylesheet looks for fields in the analyst report XML that match the values indicated in Figure 53. It then uses the information from those fields to transform the data from its original format to a HTML format as seen in Figure 54.

```
<b>Simulation Location</b>

</p><p align="left"><i><u>Analyst Considerations</u>:</i>
```

```
<font color="#00006C">This is an example of how the analyst report is generated</font>
</p><p align="left">

<i><u>Post-Experiment Analysis</u>:</i>
<font color="#00006C">The example shown is for Simulation Location</font>

</p><p align="center"><img border="1"
src="file:///C:\CVSProjects\Viskit\images\BehaviorLibraries\SavageTactics\Locations\Pearl
       HarborChart.bmp"width="640" height="480"/>

</p><p align="center"> <img border="1"
src="file:///C:\CVSProjects\Viskit\images\BehaviorLibraries\SavageTactics\Locations\Pearl
       HarborAerial.bmp" width="640" height="480"/>
```

Figure 54. HTML version of the analyst report created by applying the analyst report XSLT to the analyst report XML.

The HTML shown in Figure 54, when viewed in a web browser, produces a well formatted, subsection of the analyst report as shown in Figure 55.

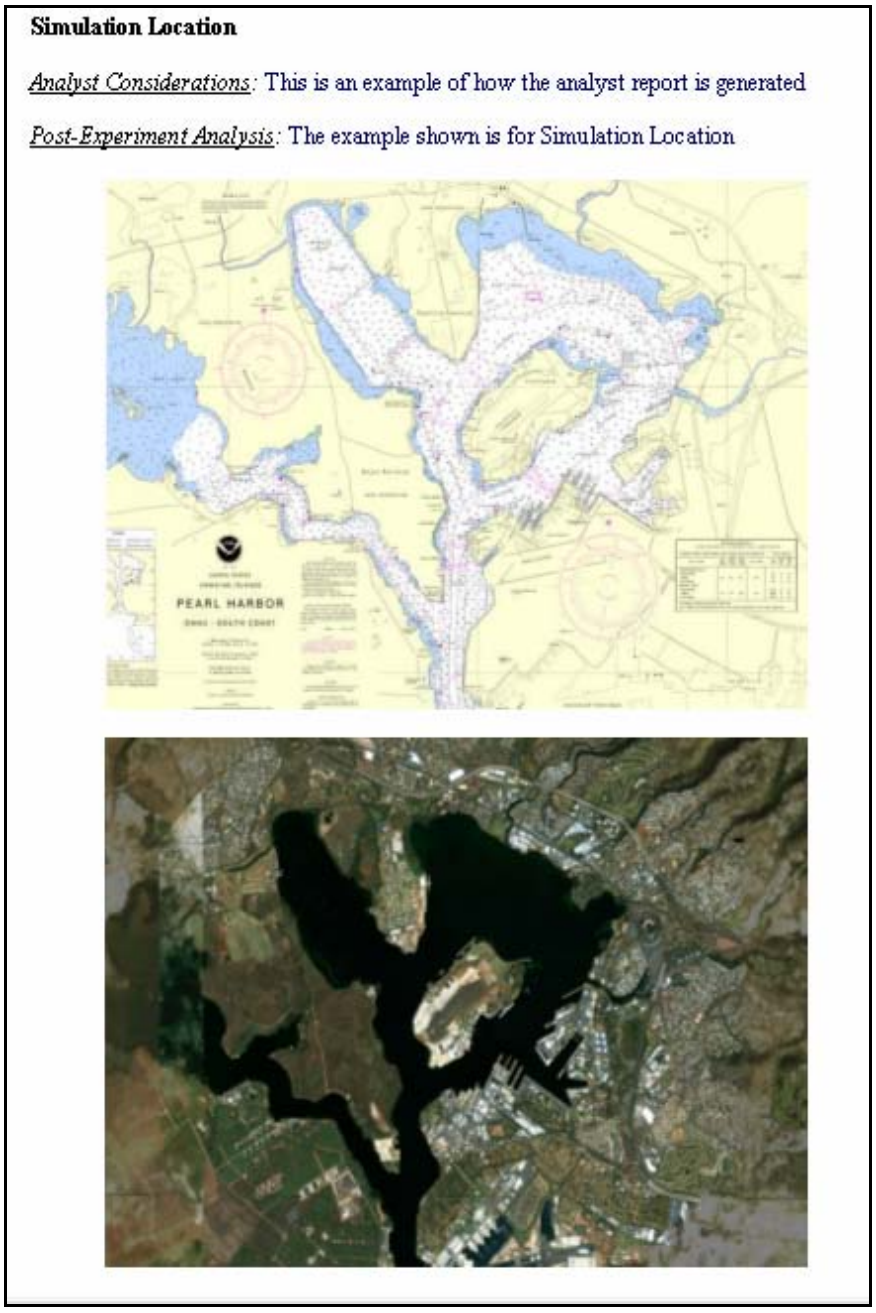
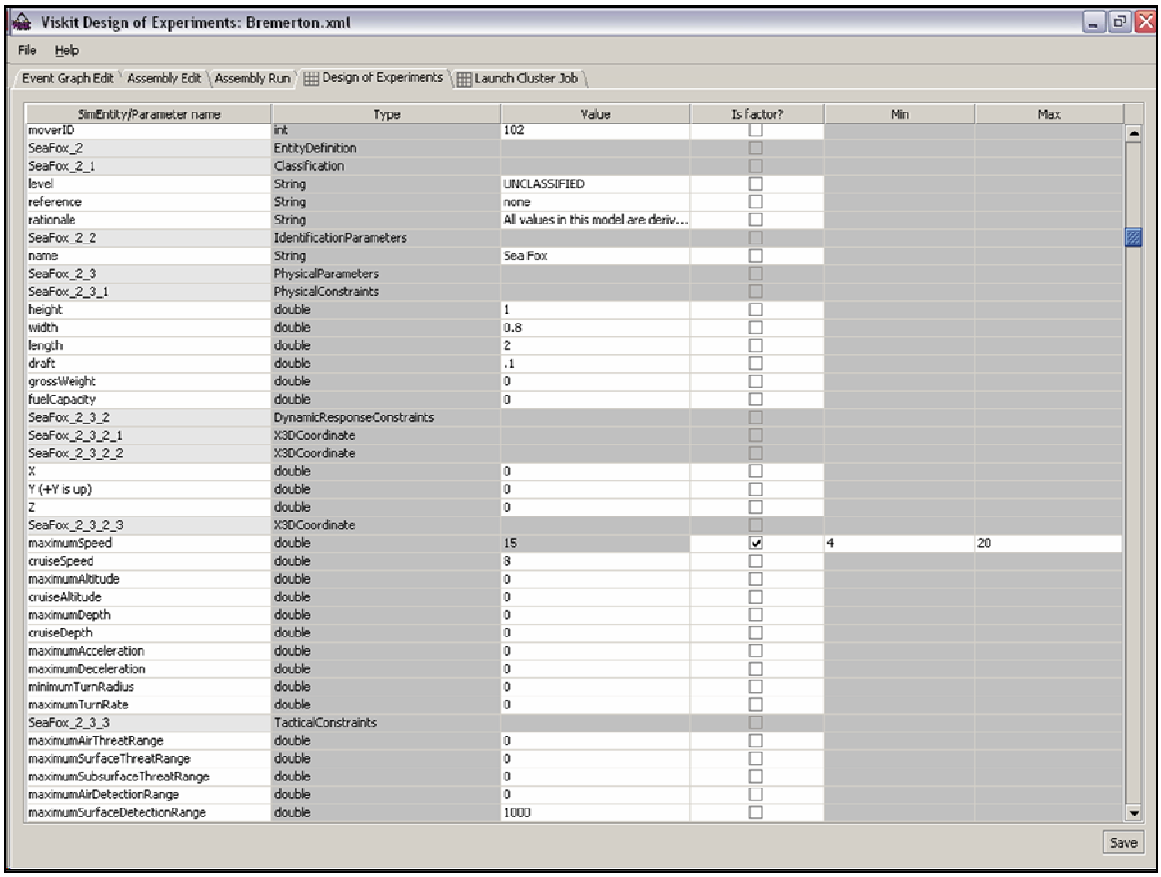


Figure 55. Generated HTML as viewed in a web browser for simulation location.

This process is used to generate the entire analyst report. Each section of the report is modifiable in a corresponding authoring panel in Viskit. Examples and descriptions of the complete analyst report interface are provided in Appendix B. Appendix C provides an example of how XSLT transformations are performed in Java applications.

F. DESIGN OF EXPERIMENTS (DOE) AND CLUSTER RUNS

Another capability of Viskit is the ability to create a complex experiment. Using an assembly file and the design of experiments (DOE) interface a user can create an experiment that can be run on a computer cluster. While a cluster is not required to do statistical analysis, this functionality allows massive replications to be run in parallel using the resources of a computer cluster.



SimEntity/Parameter name	Type	Value	Is Factor?	Min	Max
moverID	int	102	☐		
SeaFox_2	EntityDefinition		☐		
SeaFox_2_1	Classification		☐		
level	String	UNCLASSIFIED	☐		
reference	String	none	☐		
rationale	String	All values in this model are deriv…	☐		
SeaFox_2_2	IdentificationParameters		☐		
name	String	Sea Fox	☐		
SeaFox_2_3	PhysicalParameters		☐		
SeaFox_2_3_1	PhysicalConstraints		☐		
height	double	1	☐		
width	double	0.8	☐		
length	double	2	☐		
draft	double	.1	☐		
grossWeight	double	0	☐		
fuelCapacity	double	0	☐		
SeaFox_2_3_2	DynamicResponseConstraints		☐		
SeaFox_2_3_2_1	X3DCoordinate		☐		
SeaFox_2_3_2_2	X3DCoordinate		☐		
X	double	0	☐		
Y (+Y is up)	double	0	☐		
Z	double	0	☐		
SeaFox_2_3_2_3	X3DCoordinate		☐		
maximumSpeed	double	15	☑	4	20
cruiseSpeed	double	8	☐		
maximumAltitude	double	0	☐		
cruiseAltitude	double	0	☐		
maximumDepth	double	0	☐		
cruiseDepth	double	0	☐		
maximumAcceleration	double	0	☐		
maximumDeceleration	double	0	☐		
minimumTurnRadius	double	0	☐		
maximumTurnRate	double	0	☐		
SeaFox_2_3_3	TacticalConstraints		☐		
maximumAirThreatRange	double	0	☐		
maximumSurfaceThreatRange	double	0	☐		
maximumSubsurfaceThreatRange	double	0	☐		
maximumAirDetectionRange	double	0	☐		
maximumSurfaceDetectionRange	double	1000	☐		

Figure 56. The design of experiments configuration panel of Viskit.

Figure 56 shows an example of the DOE panel. In this example an experiment design point has been selected for the maximum speed value for the Sea Fox unmanned surface vehicle. The designer of the experiment is interested in what would happen if the maximum speed for the Sea Fox was varied from four to twenty knots. Viskit takes this information and creates an experiment that runs the simulation for each parameter value in the specified range. The implementation of this feature follows the nearly-orthogonal Latin-Hypercube methodology presented in (Cioppa 2002). This approach is designed to minimize the number of replications required to identify correlations between multiple

parameter combinations. The DOE implementation is an initial one. Work continues on rigorously and thoroughly implementing these important new classes of algorithms.

G. SUMMARY

Viskit has provided the means to quickly generate executable DES computer models. This process was traditionally performed by hand-coding simulations and was a time intensive, expertise dependent process. A major component of this research was to develop a repeatable and scalable methodology for creating agent-based simulations. This chapter demonstrated how Viskit was used to create discrete event, agent-based, simulations. Viskit will provide future users with a stable and reliable application and an expanding library of example discrete event simulation designs.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. DEVELOPING SCENE COMPONENTS

A. INTRODUCTION

The ability create a 3D visualization of DES was a primary motivation for many of the preceding chapters in this thesis. This chapter will discuss the approaches used in this work for generating X3D models of the environment and of agents. As graphics technology continues to evolve, many tools have been developed which greatly enhance a user's ability to create 3D models. These tools have also provided ability to perform file format conversions. This functionality has allowed the combination of multiple 3D libraries that traditionally use different formats. This chapter highlights this functionality and discusses some proven principles for 3D scene authoring.

B. HARBOR ENVIRONMENT

1. General Scenario Considerations

Perhaps the most important decision in creating a large 3D environment is the level of detail required for the model. This decision needs to be driven by the purpose of the simulation. In their survey of the utility of 3D visualization for DES (Akpan & Brooks 2005) discovered that a primary problem with using 3D models was the time and expertise required to create the scene. Survey respondents indicated that in some instances 3D models became more of a focus than the simulation driving the model. For that reason the scope of the model and the required components needs to be explicitly defined by developers of large scenarios.

For this project four exemplar locations were chosen: Indian Island, Washington; the Al-Basra Oil Terminal in the Persian Gulf; Bremerton, Washington; and Pearl Harbor, Hawaii. All four locations were created by Planet9 Studios, a project partner that specializes in creating 3D environments. The following sections provide a discussion of the level of fidelity chosen for each of these locations.

2. Indian Island, Washington

The first scenario that was developed was the ammunition pier in Indian Island, Washington. This scenario provided a simple environment that could be used to test and evaluate different components of simulation design. The ammunition pier consists of one

pier, a crane, a few buildings and low resolution terrain. Using this scene many components of both the 3D and DES models could be designed and implemented quickly since the environment itself was relatively quick to create. Figure 57 shows a scenario view of a Navy ship alongside the pier at Indian Island.

Figure 57. The Indian Island, Washington, harbor environment.

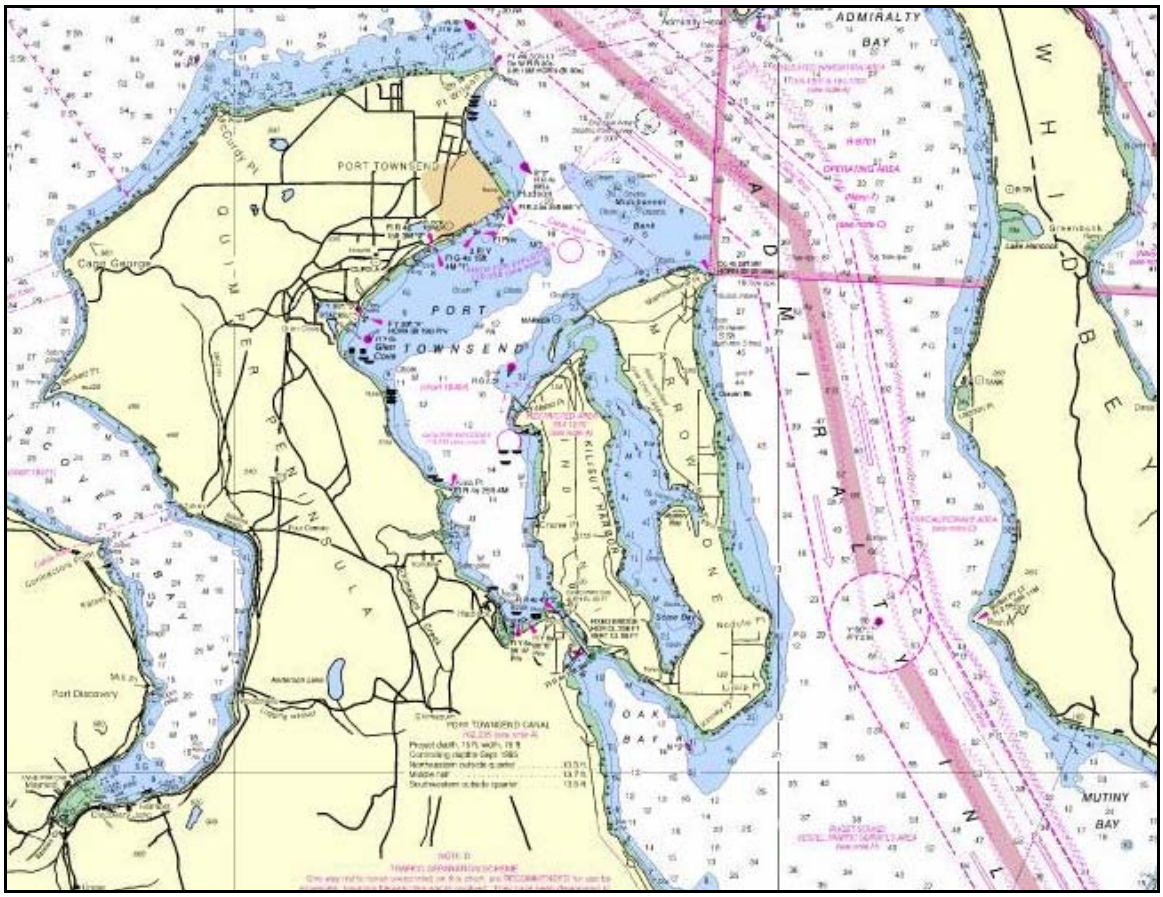


Figure 58. A nautical chart view of the Indian Island, Washington, environment (from http://www.nauticalcharts.gov/viewer/PacificCoastViewerTable.htm)

3. Al-Basra Oil Terminal, Persian Gulf

The next environment that was created was of the Al-Basra Oil Terminal (ABOT) in the Persian Gulf. Since the platform is in the middle of the Persian Gulf and there were no terrain considerations a higher level of fidelity could be afforded on the actual platform. The overall environment complexity was at the same level as Indian Island since there was only one major structure to create.

Since an oil platform is a unique structure, additional DES development and testing might be done using this model. The oil terminal model was expanded from earlier versions and was available early in the development process. Figure 58 shows a view of the ABOT model with ship models docked at the terminal and in the background. Having two example scenarios early in the development process was invaluable for developing agent behaviors. The differences in these environments provided unique behavior design challenges which added to the variety of the behavior library.

Figure 59. The Al-Basra Oil Terminal scenario environment

4. Bremerton, Washington

The next environment modeled was Bremerton, Washington. In this environment terrain was developed at a higher level of fidelity and included a larger area. Additionally more buildings were modeled and the piers and waterfront area were developed with a greater level of detail. This environment was primarily used to develop agent behaviors in a complex waterfront environment. Bremerton was also used to determine the best way to model and represent various types of pier side harbor equipment and personnel. Figure 59 shows a simulation running in the Bremerton environment.

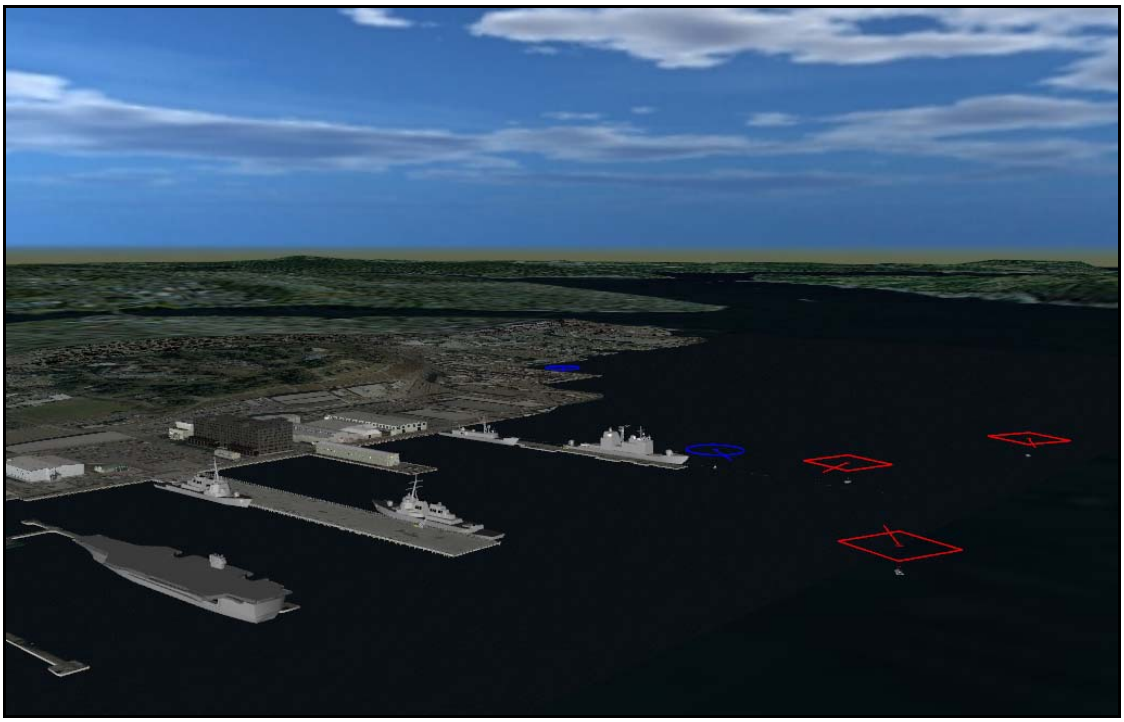


Figure 60. The Bremerton, Washington, harbor environment

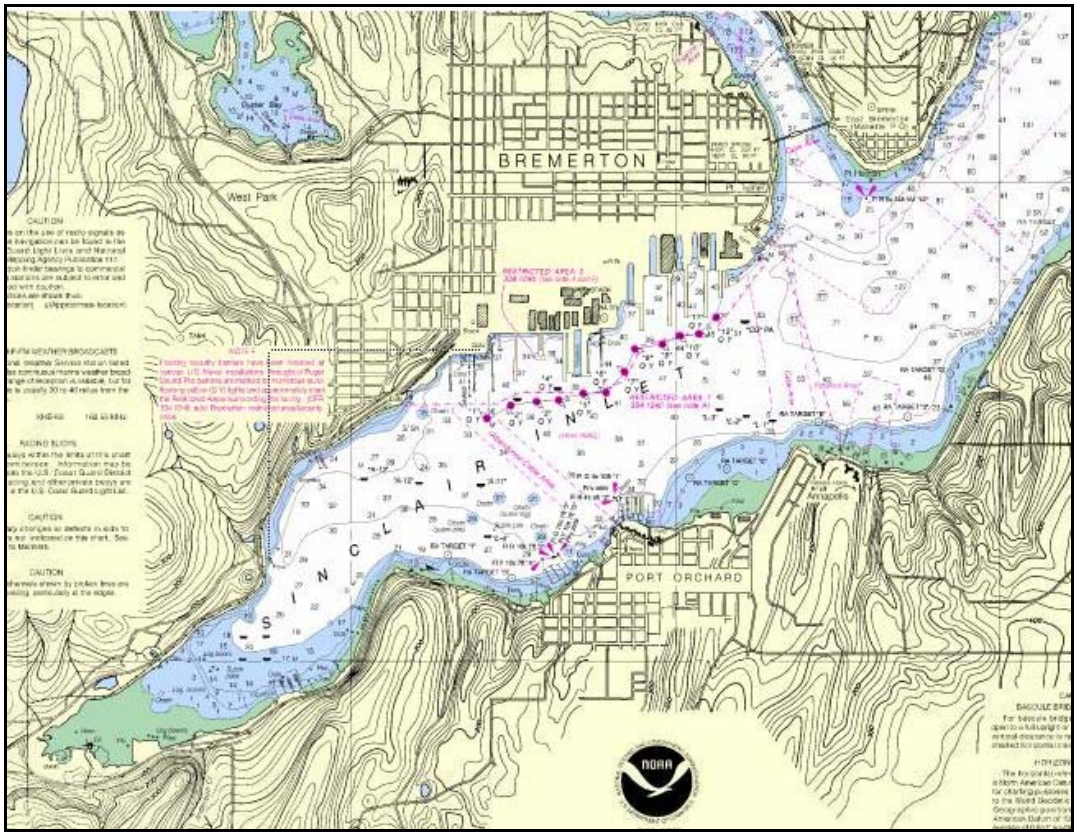


Figure 61. A nautical chart view of the Bremerton, Washington, environment (from http://www.nauticalcharts.gov/viewer/PacificCoastViewerTable.htm)

5. Pearl Harbor

The final environment created for this project was Pearl Harbor. This model was created by expanding earlier models of Pearl Harbor that Planet9 had developed. The goal of this scenario was to model a complete harbor environment and combine many of the concepts and approaches developed using the other environments. The amount of time required to create this scale of a model was much longer than the three previous scenes. Pearl Harbor was created after three months of work while the earlier scenes were constructed between one to four weeks. A detailed description of the modeling process for the Pearl Harbor scenario is provided in Chapter VIII.

Figure 62. The Pearl Harbor, Hawaii, harbor environment

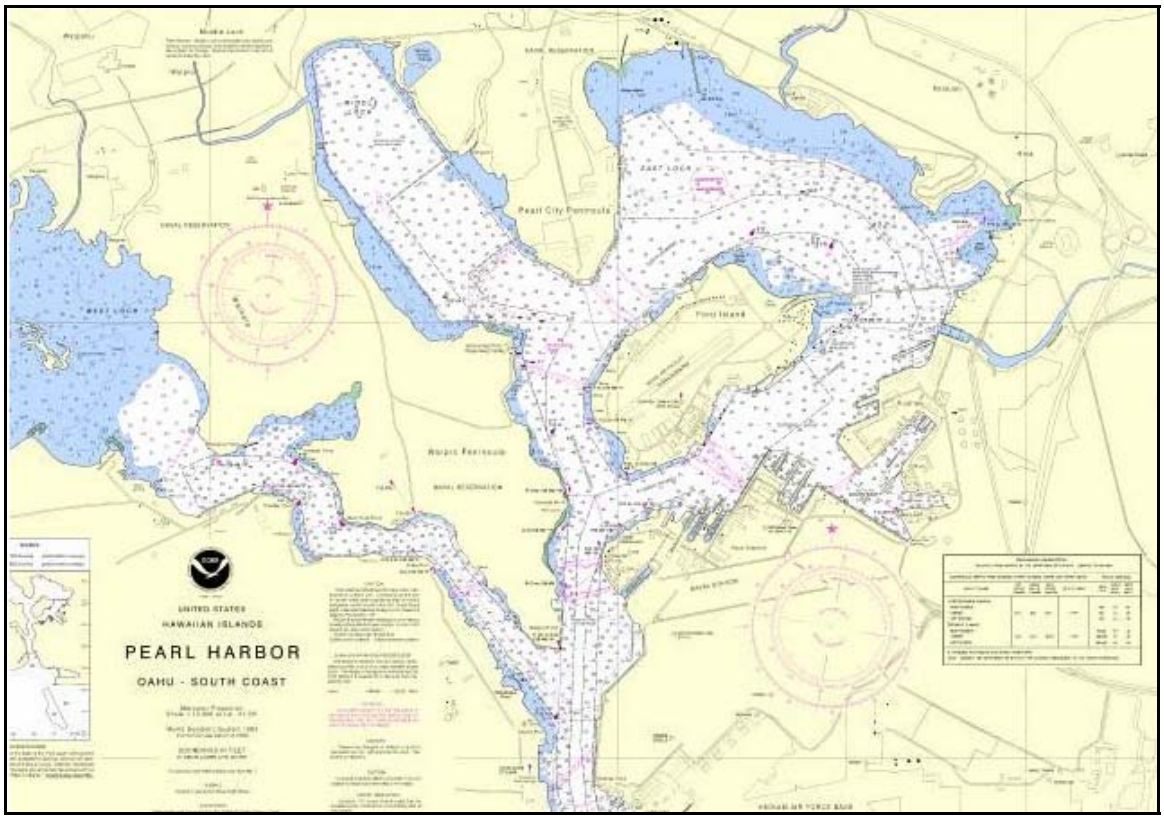


Figure 63. A nautical chart view of the Pearl Harbor, Hawaii, environment (from http://www.nauticalcharts.gov/viewer/PacificCoastViewerTable.htm)

C. MODELING ENTITIES

1. Leveraging Model Libraries

(Harney 2003) discusses the importance of leveraging existing 3D model archives when developing a scene. Perhaps the easiest way to minimize the time required to create a 3D visualization is to utilize models that already exist in repositories such as the SAVAGE 3D model archive. 3D authoring tools allow the inclusion of models from other repositories in cases where a 3D model existed but is not in the SAVAGE archive. Delta3D, an open source game engine developed at NPS has an open source asset library. The models used by the Delta3D group are normally in 3D Studio (3DS) graphical format. Vizx3D has a file conversion utility that converts 3DS files as well as many other file formats to X3D.

When the ABOT scenario was created background commercial shipping traffic was identified as a simulation requirement. While SAVAGE had some models of this type, Delta3D had a number of models that were created for a shipboard navigation trainer.

Figure 61 shows the result of converting the 3DS file to X3D. The ability to convert file formats has opened new doors for the sharing and integration of models in a variety of formats. More information regarding Delta3D can be found at www.delta3d.org (accessed August 2006). The Delta3D asset library can be downloaded at https://sourceforge.net/project/showfiles.php?group_id=113203&package_id=143034 (accessed September 2006).

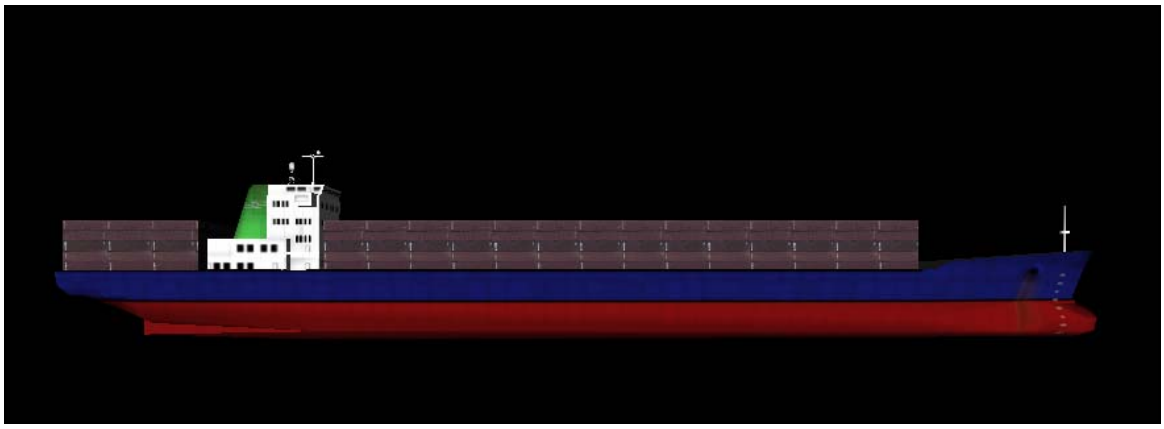


Figure 64. A container ship model displayed in X3D form after being converted from the Delta3D asset library.

2. Creating New Models

Many models needed for this project did not previously exist in either repository. Over forty models were created for this project including harbor equipment, buoys, watercraft, weapons and overlays. In all cases the same authoring process was used. The process of creating 3D models in X3D format has been greatly enhanced by the tools listed in Chapter II. For each model a complex geometric representation was first created in Wings3D. The file was then converted to X3D format using Vizx3D and detail was added to the model. Finally, the file was saved and validated using X3D-Edit, another open-source X3D authoring tool (Brutzman 2002). The following sections show how each step of this process was performed for the creation of a harbor security boat.

3. Wings3D Mesh Editor

After the need for a model was identified the first step in this approach was to use Wings3D to create a base model. Wings3D allows for the easy creation and manipulation of complicated wire frame meshes. When exported into VRML these

meshes are transformed into the X3D equivalent Indexed Face Sets. Figure 62 shows the completed security boat model as displayed in Wings3D. This model, with tool familiarization, took about an hour and a half to create. The power of this tool does have a potential pitfall. Wings3D will allow an author to create as complicated a model as he desires.

When exporting this model there is a possibility that the geometry created is so complex that a simulation could be bogged down when trying to render the model as part of an entire scene. As with scenario creation it is important to determine the level of fidelity required.

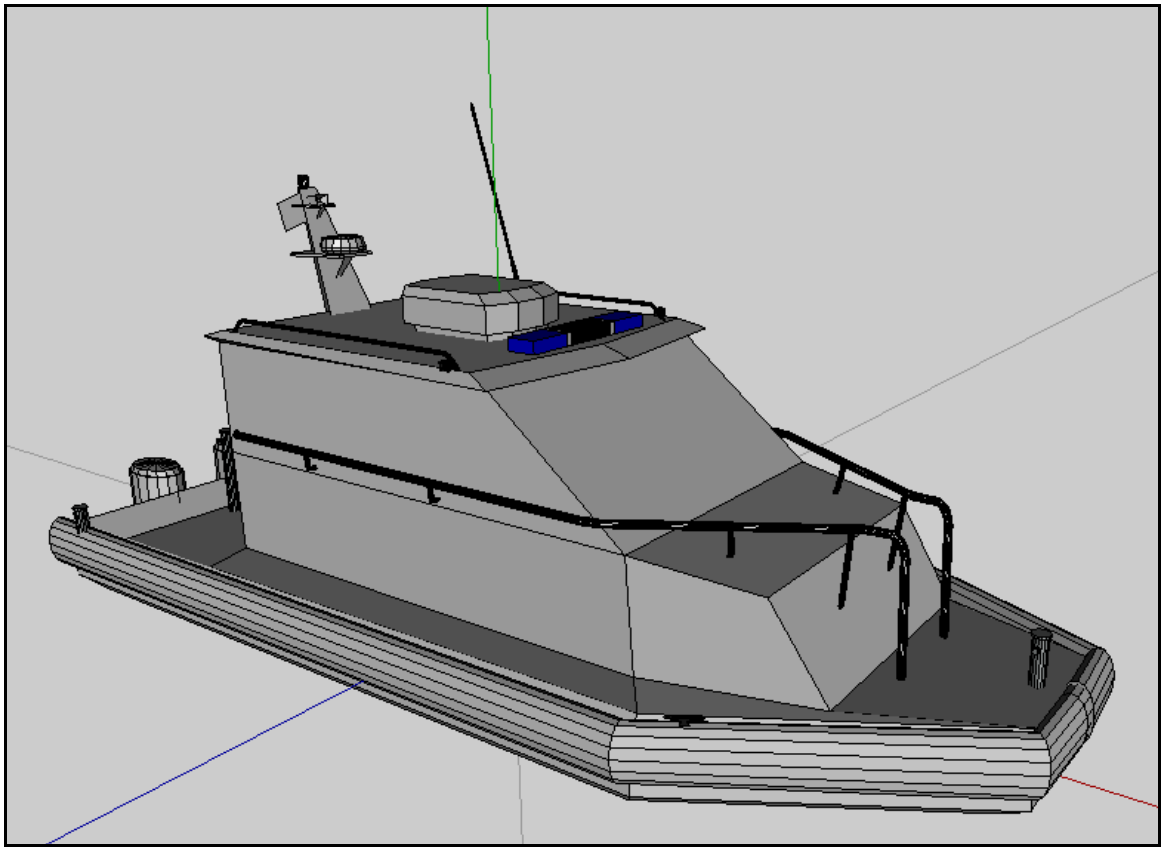


Figure 65. Early modeling of a security boat using the Wings3D authoring tool.

After the base model was created in Wings3D it was exported into a VRML formatted file. While Wings3D does have the ability to add color and other details to a model there are multiple differences between the format outputted and the desired X3D format. As a result Wings3D was only used to create the overall geometry. After the

model is completed the resultant VRML model looks like the model depicted in Figure 63.

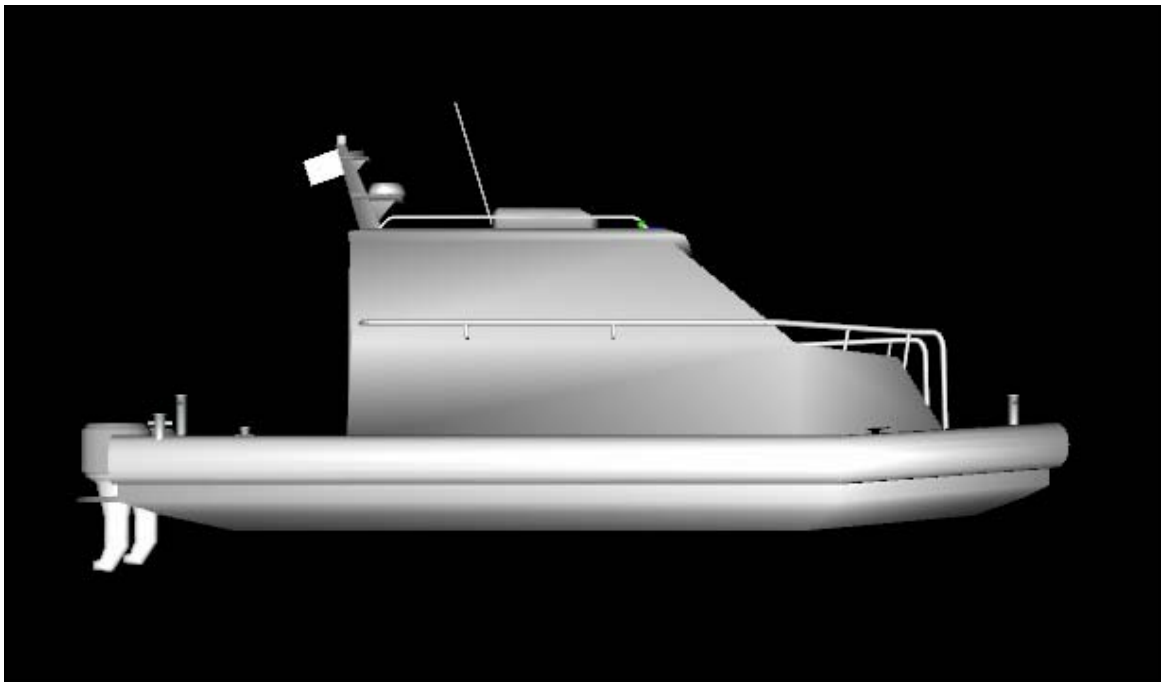


Figure 66. Security boat model in VRML format after being exported from Wings3D

4. Vizx3D Scene Graph Authoring Tool

Vizx3D was used to add detail and other X3D components to the basic geometry model exported from Wings3D. This detail includes textures, colors, lights, animation and other scene components. Using Vizx3D ensured that textures and other detail components would be created in a proper X3D format. The completely detailed security boat model is shown in Figure 64.

Figure 67. Security boat after applying details and textures using Vizx3D authoring tool

By using Vizx3D the amount of time required to make 3D models was greatly reduced. More importantly, since Vizx3D is an X3D authoring tool the file can be saved as an X3D file and the potential for file conversion inconsistencies is eliminated. The ability to use multiple tools to quickly create 3D models is very useful; however, every time multiple tools are used it is important to test for inconsistencies when conducting file conversions.

5. X3D-Edit Scene Validation

The final step in the model authoring process was to ensure that the generated model was in a consistent format. The X3D authoring best practices guide (Brutzman 2006) lists the structure used for models in the SAVAGE archive. This format provides a uniform representation of how the model should be formatted before being added to the archive and was used for all models in this project.

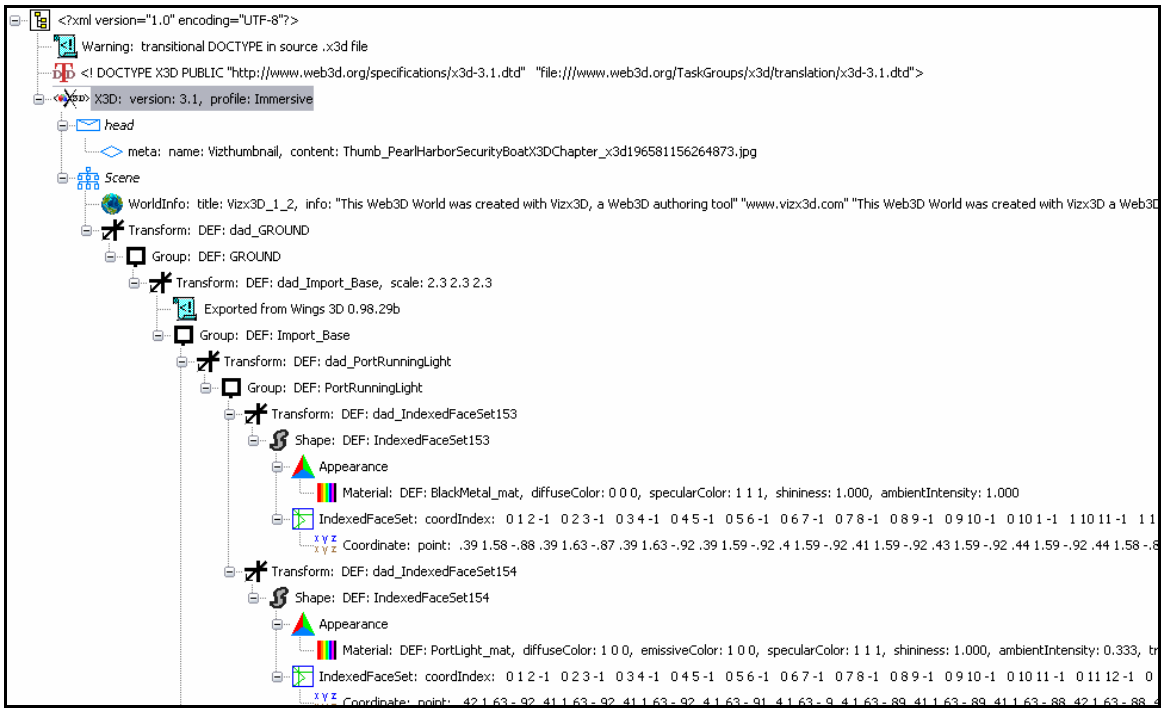


Figure 68. X3D-Edit representation of the security boat model after exporting and processing in Wings3D and Vizx3D authoring tools

Figure 65 shows the format of a model that was created in Wings3D, modified in Vizx3D and saved as an X3D file. While this model will be displayed in an X3D browser the format does not follow the best practices guide mentioned earlier. Figure 66 shows how the file header was changed to a clean and concise format.

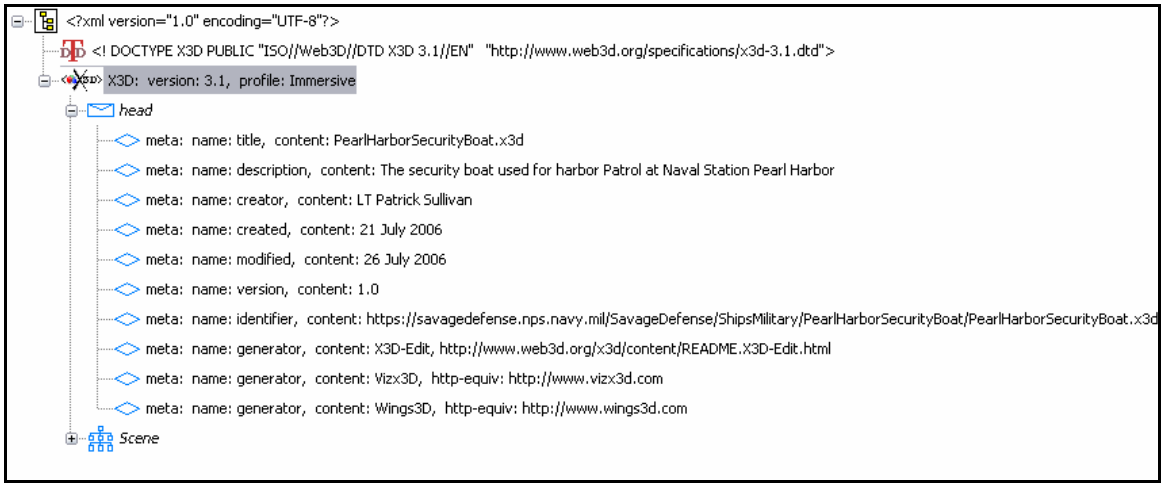


Figure 69. Modified header of the security boat model following X3D authoring best practices guidelines provided in (Brutzman 2006).

In addition to a modified header, additional information can be added to the model. (Rauch 2006) outlined how to include SMAL Entity Definition information inside of the body of a model. Figure 67 shows part of the SMAL Entity Definition information for the security boat model.

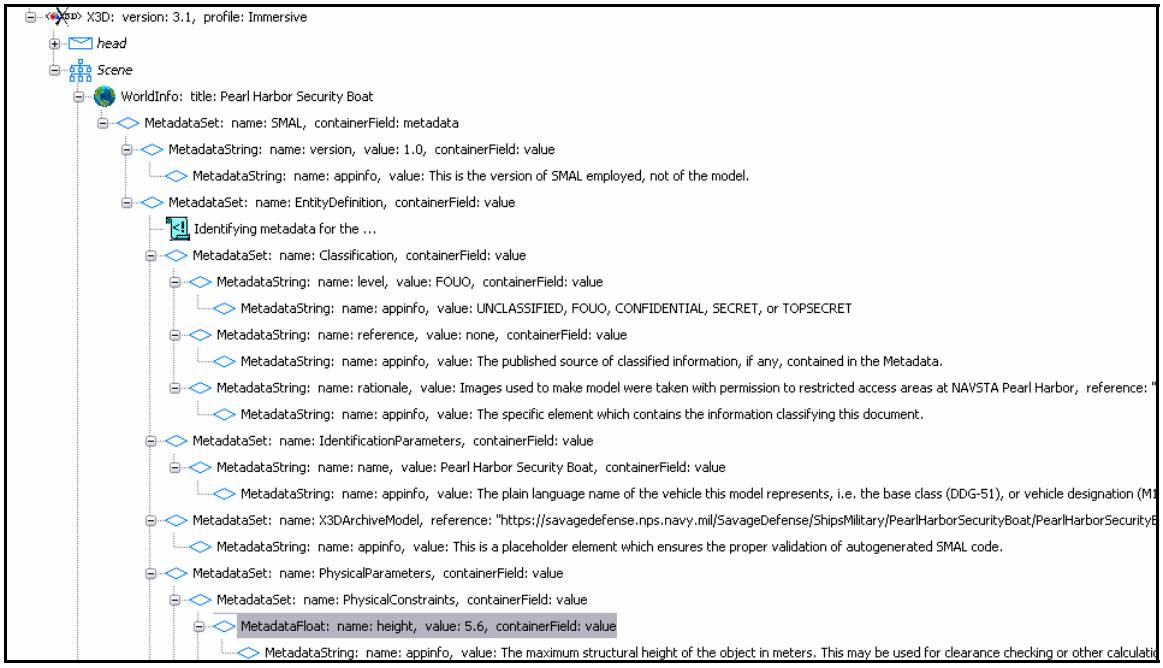


Figure 70. Security Boat model with SMAL metadata embedded providing detailed information about the model.

In the above example the highlighted field is for the height of the entity. The value for this field can be provided as part of the model. As a result the model has more information than just the graphics components and the header information. This information can be used by tools such as Savage Studio, discussed in Chapter VIII, to autogenerate Viskit based simulations.

After modifying the model file in accordance with the best practices guide the model is checked against the X3D specification utilizing the validation process of X3DEdit. X3DEdit verifies the content based on an XML schema that is defined by the specification. If all fields are properly used and there are no specification violations then the model is ready for use in the archive. If not warnings or errors are provided in the X3D Edit console notifying the author of possible issues that should be addressed.

The process of properly formatting an X3D model and validating it against the specification serves two purposes. First it ensures that the use of models for a specific project can be reused by any individual using the SAVAGE archive. Second it ensures that any formatting issues resulting from using various tools for model creation, conversion, and detailing have not created a model that violates the X3D specification.

6. Model Prototype Implementation

X3D has well over fifty node types for creating 3D objects and worlds. X3D also has the ability for users to create their own node types. This is accomplished by using a prototype definition (PROTO). In this project PROTOs were used to combine models and minimize the complexity of the harbor environment. A detailed description of prototype definitions can be found in (Ames, Nadeau, & Moreland 1997).

To illustrate how this was used we will look at the creation of the buoy prototype. For this project three buoy types were created: a mooring buoy, a red buoy, and a green buoy. In the Pearl Harbor scene, fourteen buoy models exist in the environment. Rather than having three different model references in the Pearl Harbor scene it was desired to have one centralized buoy representation that offered an author the opportunity to select among the three choices.

This was done through the creation of a buoy prototype. Figure 68 shows the buoy prototype that was created.

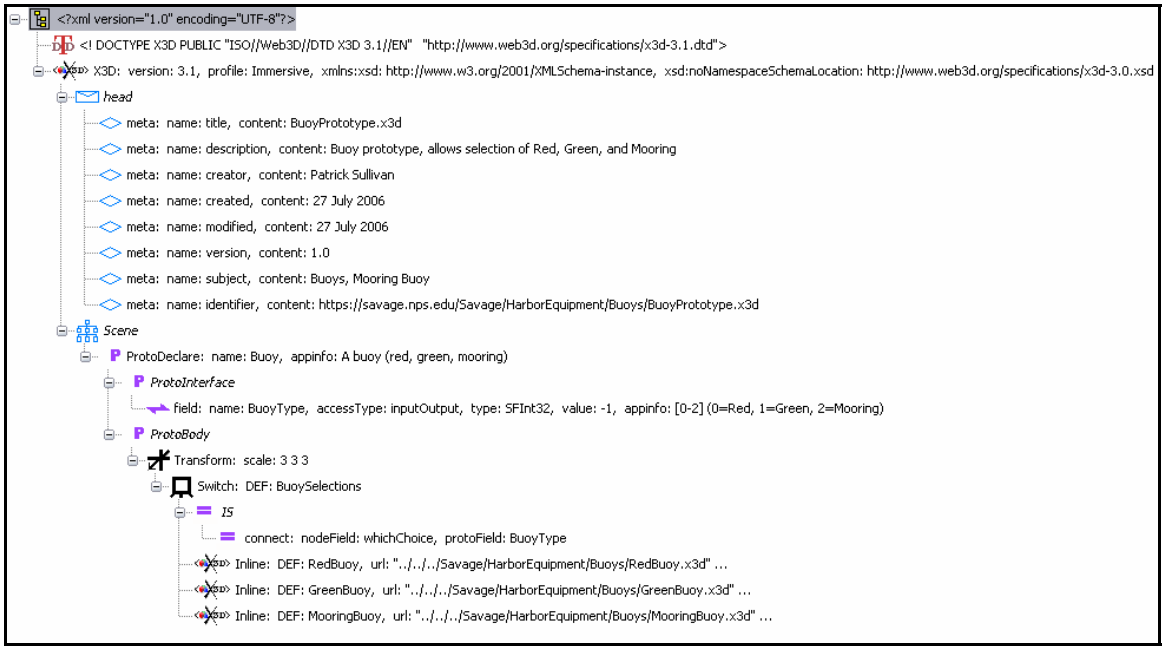


Figure 71. Prototype example depicting a prototype for buoys.

When a prototype is declared a name and description is provided for reference. The next part of the prototype is the proto-interface which indicates the fields that should be populated by a user of this prototype. Finally there is a proto-body which includes all of the components of the prototype model as well as references to the user's inputs for integration. In this example a user is able to enter a number from 0-2. This value corresponds to a buoy selection. The value that is entered by the user is used to set the 'which choice' field of the buoy selection switch node. The effect of this construction is that a user can specify a buoy by simply providing an integer entry.

After the prototype is created and saved, other scenes can create a usable copy of the prototype as shown in Figure 69. This example shows that a reference to the location of the prototype file is provided as well as the required interface fields.

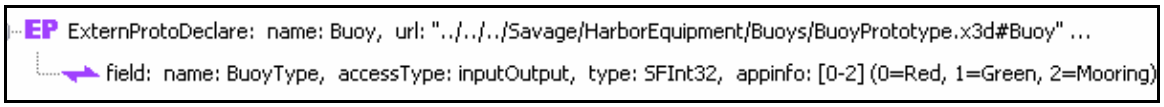


Figure 72. External Prototype declaration of the buoy prototype.

Once this reference has been created locally, instances of the prototype can be created and used anywhere in the scene. Figure 70 shows some of the instances of buoys from the Pearl Harbor scenario.

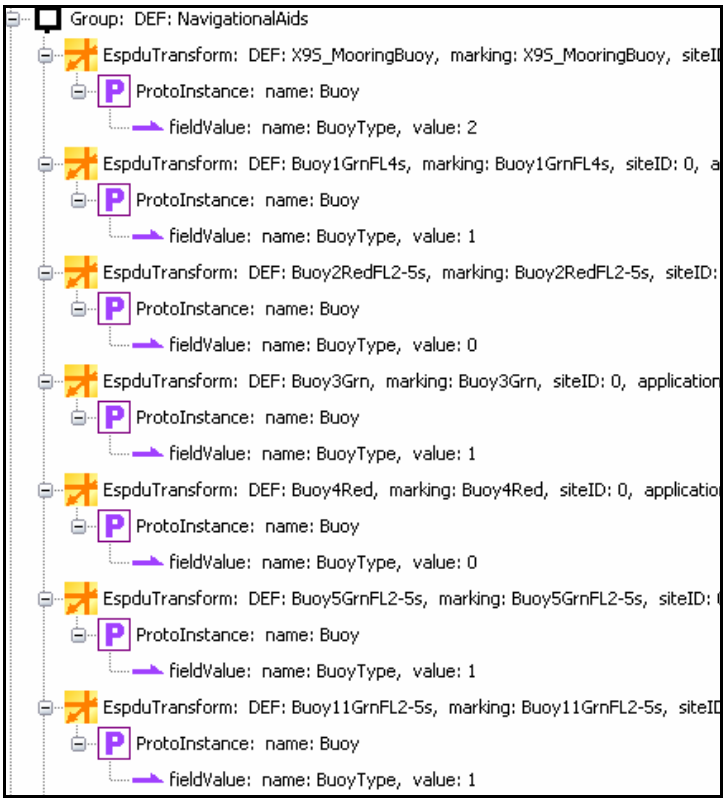


Figure 73. Buoy prototype instances in the Pearl Harbor Scenario

Notice that for each proto instance only the selection number is used. Using this prototype multiple buoy types were easily created and placed in the harbor environment. Figure 71 shows an example of buoys as displayed in the Pearl Harbor scene.

Figure 74. Three instances of the buoy prototype as displayed in the Pearl Harbor scene.

While the buoy prototype example is rather simple the prototype concept was also used for other scene components. Figure 72 shows the use of a prototype for a ship's brow watch. The prototypes used in this image include where a ship's brow will be located, weapon types and locations for a ship, the color of the canopy on the pier, and the name and crest of the ship. All of these components are combined into a prototype definition used for adding detail to the pier watch model.

Figure 75. Example of a complex prototype being used for multiple details of a pier watch model.

7. Design Pattern for the Scenario Scene Graph

The final section of this chapter discusses the design pattern that was used for combining X3D models to form a scenario scene graph that provides a 3D visualization of a running DES.

When creating an X3D file that will use the DIS protocol an author must ensure that the model will be capable of receiving and using DIS information. Figure 73 shows the header of the Pearl Harbor scenario file. Note that the first line in the header for this file indicates that it is using the DIS protocol. Without this component node an X3D file will not be able to process DIS information. Also note that the first elements in the Scene are External prototype declarations. By declaring these prototypes early in the scene it is clear to future users what various prototypes are available in the scene graph.

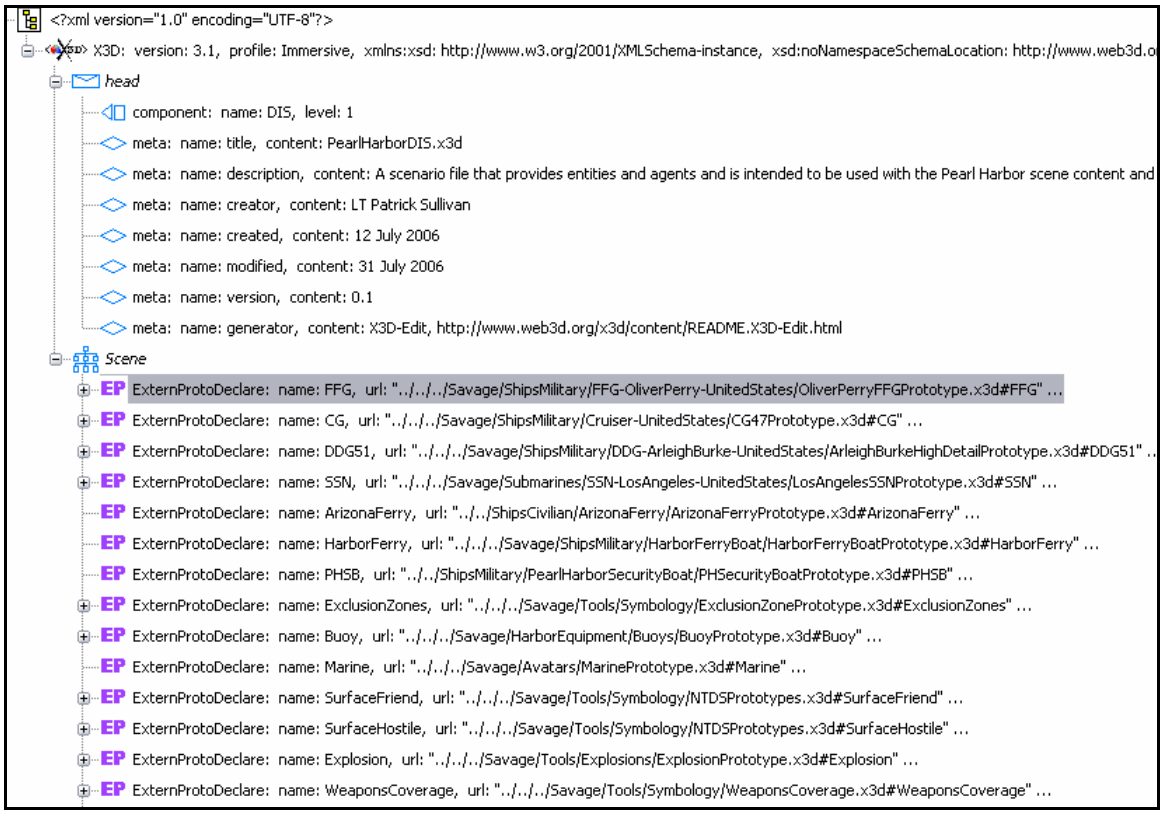


Figure 76. Simulation design pattern used for structuring X3D for the Pearl Harbor Scenario

Figure 74 shows the bottom half of the scenario file. The convention used for this project was to follow the external prototype declarations with scene navigation information. Browser viewpoints and individual models follow the navigation node of the scene. This pattern is a style preference but authors should be consistent when creating multiple scene graphs so that they can be easily referenced by future users.

Figure 77. Grouping of like models in the Pearl Harbor scenario X3D file.

The other critical component of a scene graph required to implement the DIS protocol is the Entity State Protocol Data Unit (ESPDU) transform. The ESPDU transform listens to a multicast socket connection to receive information regarding the state of its entity. This information includes position information which is used to move the model once the information is received.

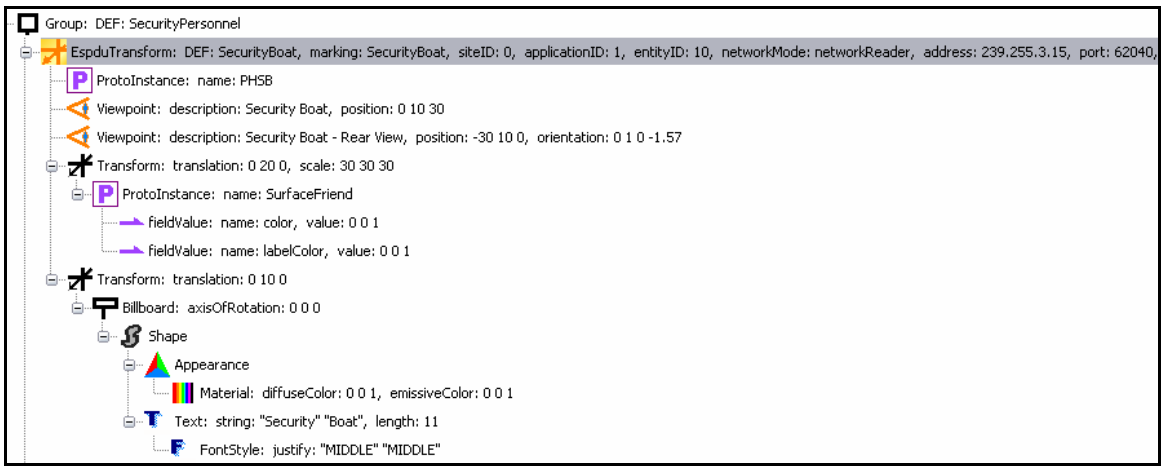


Figure 78. Example use of the Entity State Protocol Data Unit (ESPDU) transform node in the Pearl Harbor scenario.

Figure 75 shows the ESPDU transform for the security boat entity in Pearl Harbor. There are many important components to this object that must have values for the transform to receive updates. The fields that must be completed are the site identification number, the application identification number, a multicast address, and a port number. All of this information is also required for the Scenario Manager SimEntity Node in Viskit as shown in Figure 76.

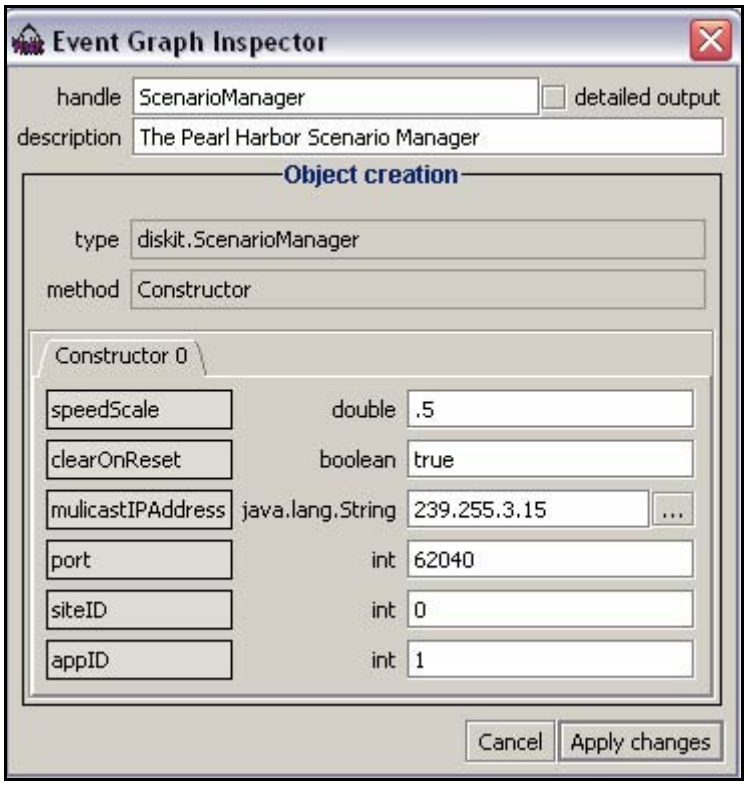


Figure 79. Pearl Harbor scenario manager panel showing the ESPDU transform connection information as required parameters.

All entities in the same scenario must have the same values for these fields in a parent ESPDU transform object. The last field in the ESPDU that must be populated is the entity's identification number which must be a unique number. This number is represented by the mover ID number that is required for all 3D mover objects in Viskit. This connection allows 3D visualization of the simulation. Viskit creates packets for all moving entities at the time interval specified. The result is a 3D visual display of the environment that corresponds to the DES model.

D. SUMMARY

This chapter provided a discussion of the specific 3D modeling issues for this thesis. While 3D visualization is an important part of the overall simulation it should not take priority over the underlying DES model. Since the simulation and the visualization are decoupled both can be worked on simultaneously and one should not prohibit the other's development. This ensures that a good, running simulation is created with or without a 3D display. The tools mentioned in this chapter have greatly reduced the amount of time required to create high-quality 3D models. By using tools of this nature and adhering to proven 3D modeling practices, the amount of time required to generate a complete scenario environment is likely to remain reasonable and feasible.

VII. SAVAGE STUDIO SCENARIO AUTHORING TOOL

A. INTRODUCTION

The user interface of Viskit has provided the ability for non-programmers to create sophisticated DES models. Savage Studio is an interface designed to provide users with the ability to generate a complete scenario with a DES and 3D graphics representation. This tool will fully leverage the underlying XML architecture of Viskit and X3D graphics to autogenerate a complete simulation. The primary motivation for the creation of SMAL (Rauch 2006) was to provide an architecture that could bridge the gap between 3D graphical models and other simulation applications. The vision of that work is realized in Savage Studio where SMAL serves as the connection between the model libraries of the SAVAGE model archive and the behavior libraries of Viskit.

B. INTUITIVE INTERFACE FOR SCENARIO AUTHORING

The Savage Studio interface is designed to provide a user with the ability to setup simulation components by adding them to the scenario in a drag-and-drop panel. Figure 77 shows the main panel of the Savage Studio graphical user interface.

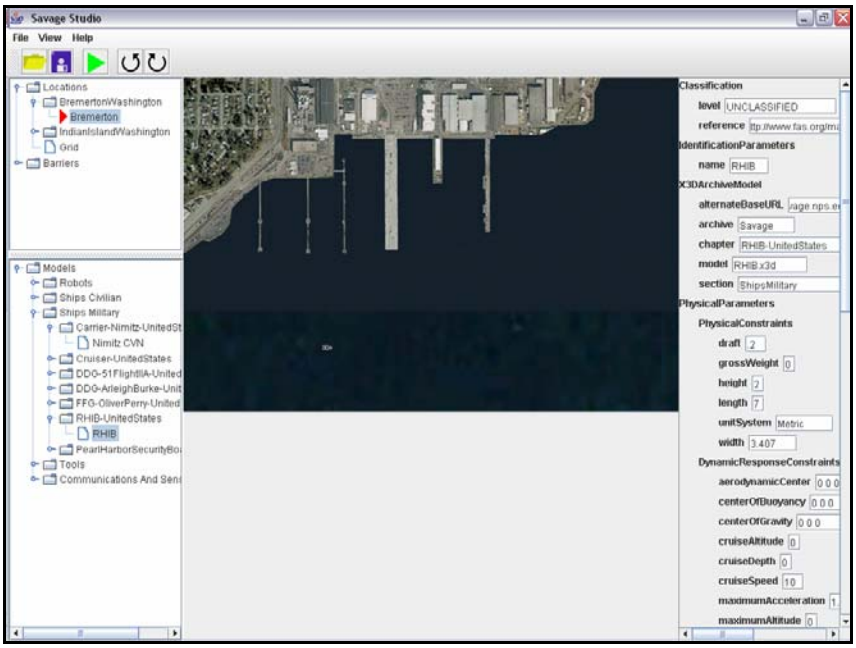


Figure 80. Depicts the Savage Studio scenario authoring graphical user interface

This interface allows users to select models to place into a scenario. The user can then update or change parameter values for a model and define a specific behavior definition for the model. Once all of the components of the simulation have been selected Savage Studio creates a Viskit assembly file that connects the simulation components. Additionally, Savage Studio creates a 3D model of the scenario which corresponds to the DES model. The simulation can then be executed by running the DES model in Viskit from the Savage Studio interface.

The functionality provided by this interface significantly reduces the amount of time required to create and run a scenario. The remainder of this chapter examines how Savage Studio integrates the DES model of Viskit with the X3D graphics components.

C. AUTOGENERATION OF 3D MODELS

All of the X3D scenario scene graphs created for this project were authored in separate applications. Savage Studio provides the ability to select various model components and combine them to create a 3D representation of a scenario. The models that are available for use in the scenario are from the SAVAGE model archive. Figure 78 shows the model selection panel from the Savage Studio GUI.

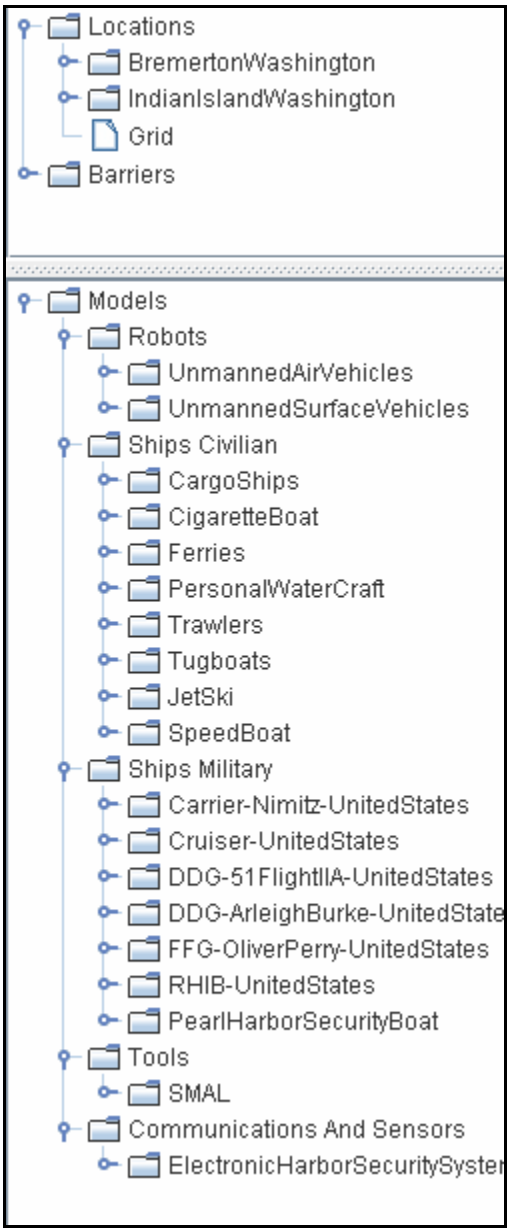


Figure 81. Depicts SAVAGE archive models that can be used by Savage Studio

Users can select models that are included in this list and add them to the scene. After all of the components are selected Savage Studio creates a complete 3D scenario file that is properly formatted to receive DIS information packets and display the DES simulation. Figure 79 shows a scene that was autogenerated by Savage Studio.

Figure 82. Shows a 3D scene generated by Savage Studio

The models of the SAVAGE archive that can be used in this interface are those that have SMAL metadata inside the body of the model. This ensures that the 3D model also includes specific information that can be used by Savage Studio to create a Viskit objects.

D. LEVERAGING SMAL METADATA

A SMAL metadata set contains specific parameter information about an object and is embedded in the X3D representation of that object. The combination of 3D graphical components and parameterized metadata ensures that the model can be rendered and viewed in a 3D browser, and that Savage Studio is able to represent the model in a Viskit assembly file. Figure 80 shows the parameter panel of the Savage Studio GUI. In this example the SMAL metadata in the 3D model has been imported giving a user the opportunity to review and modify the parameters for a model.

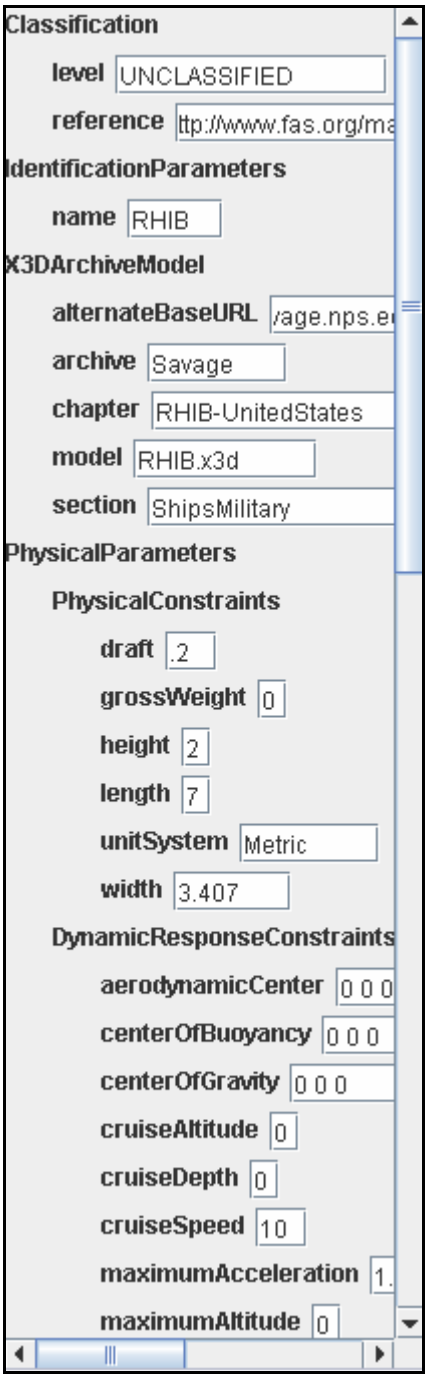


Figure 83. Depicts the entity parameter panel of Savage Studio

Savage Studio uses this parameter information to create the Viskit assembly file. Allowing parameter changes ensures that the user has the same level of control over scenario authoring as they do with Viskit application alone.

E. AUTOGENERATION OF VISKIT COMPONENTS

Savage Studio creates a Viskit assembly file using the parameter values for each model. Figure 81 shows an example assembly file that was generated using Savage Studio. As users select models and add them to the scenario, representations of those models are also created as SimEntity nodes. In Figure 81 a RHIB model is created that is using the Military Patrol Craft behavior event graph.

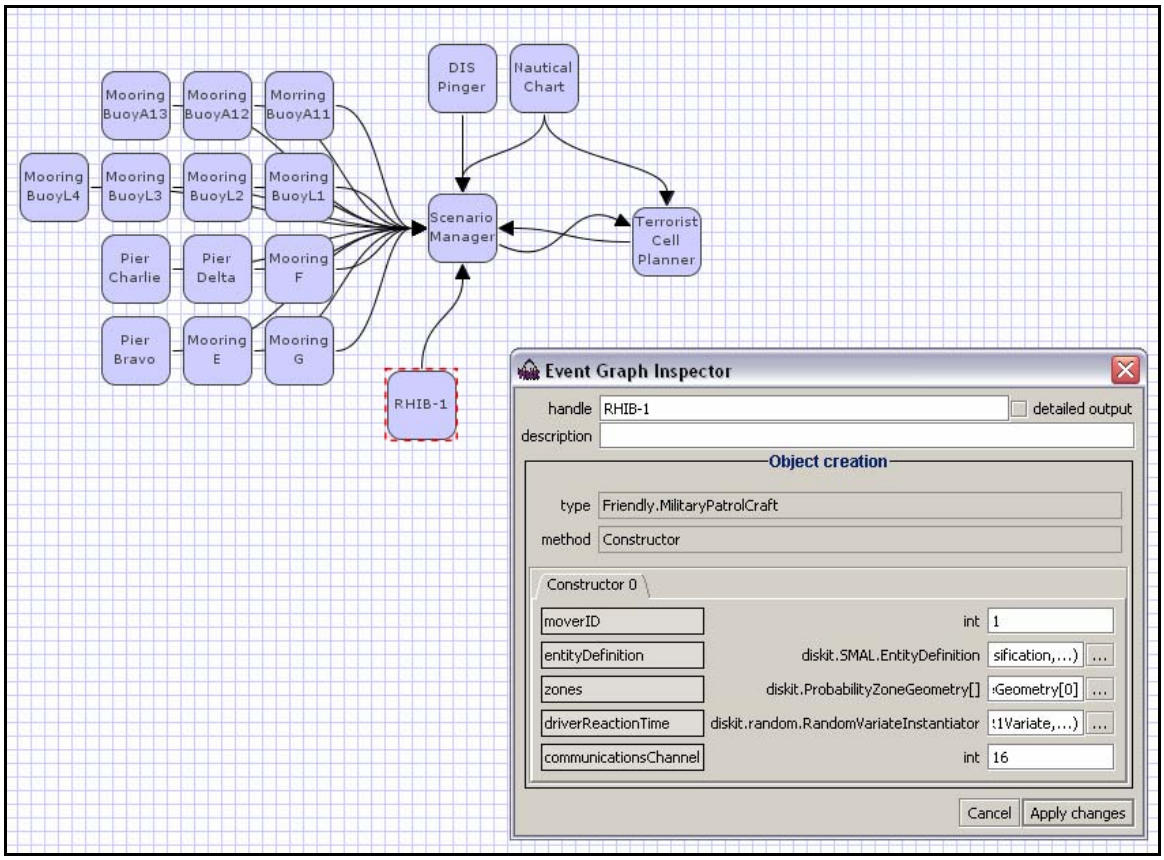


Figure 84. Viskit assembly file that was autogenerated by Savage Studio

Savage Studio uses a baseline location assembly file to represent the harbor environment. For the scenarios in this project this assembly file includes all of the objects in the simulation and the nautical chart object. The remainder of the assembly is populated by Savage Studio as the user selects more entities for a simulation run. The example in Figure 81 shows the parameter entry panel for a RHIB model. The

parameters for the model are automatically populated and the required assembly connections created by Savage Studio when the scenario is processed.

F. SUMMARY

Savage Studio provides a user with the ability to create a complete simulation with minimal setup and design. Savage Studio does require that a location has been modeled in both Viskit and in X3D. Once the development of those two components is complete, users with basic computer skills can use Savage Studio to create and run a full simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. FULL HARBOR SCENARIO — NAVAL STATION PEARL HARBOR

A. INTRODUCTION

The final phase of this research was to create a large-scale harbor environment that incorporated the components necessary to evaluate waterside AT/FP measures. During this phase, the goal was to identify some of the challenges associated with scaling a scenario to a full harbor environment and establishing the recommended approach for using and expanding the behavior libraries for a specific real-world location. This chapter presents the creation, design, and implementation of the Pearl Harbor scenario to illustrate how a scenario of this scale is constructed.

B. SCENARIO DEVELOPMENT

1. Problem Definitions

Prior to working on new behaviors for this scenario, a problem definition was created listing the minimum requirements for the simulation. This definition was used to determine what functionality would have to be added to the existing behavior libraries to simulate the harbor environment. Table 6 shows the major components of this definition statement.

Component	Description
Detailed Waterfront Area	The entire waterfront area had to be represented including piers and navigational aids
Additional Personnel	Shipboard watch standers, pier rovers, and the Pearl Harbor Control Tower had to be added
Harbor Traffic	Craft that were specific to Pearl Harbor were created and simulated
Exclusion Zones and Communications	Exclusion zones had to be represented and communications had to be expanded for multiple radio circuits
Security Boat Patrol	New implementation for patrolling an entire harbor area by a centralized harbor patrol boat

Table 6. Special Design Considerations for creating the Pearl Harbor Scenario

The items listed in Table 6 were identified as the minimum required components for a simulation to be a realistic representation of the harbor environment. The specific requirements outlined above were implemented by using the approach for designing the three lower-fidelity scenarios and by expanding the existing behavior library. The following sections discuss the implementation of this approach.

2. 3D Model Configuration

As mentioned in Chapter VI, the amount of effort required to create a 3D model of Pearl Harbor was much greater than the effort for creating the original 3D environments. To begin this process, the author and a professional photographer from Planet9 Studios traveled to Pearl Harbor to capture the imagery required to create the 3D model. During this five-day effort the team was provided access to the majority of the installation and given a two-hour tour of the waterfront on an off duty security boat. The waterfront tour enabled imagery to be captured from the waterfront perspective.

The next phase of development required Planet9 to create the buildings, piers, and other structures of the harbor. This effort took three months to complete. While the high-fidelity model was being constructed, lower-fidelity versions were provided to the author to use for testing and evaluating agent behaviors. This parallel design process allowed the 3D model development and agent behavior development to occur in parallel.

Three additional 3D models were needed for this scenario. Pearl Harbor has a Navy owned harbor ferry, a ferry that is used for transporting visitors to and from the Arizona Memorial, and a security boat. The security boat, used as an example in Chapter VI, and the harbor ferry shown in Figure 82 were created using imagery taken during the photo trip to Pearl Harbor. The Arizona Ferry shown in Figure 83 was created by Planet9 studios.

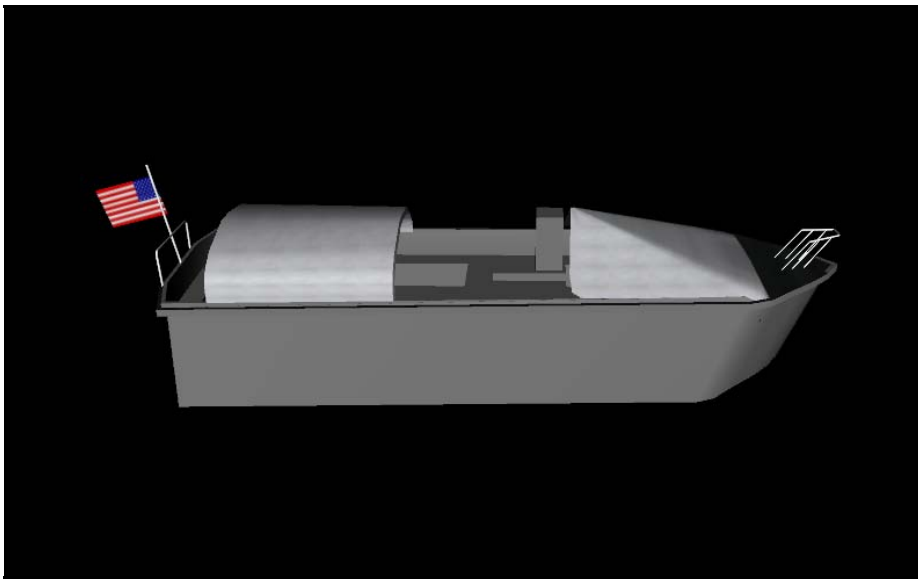


Figure 85. The Pearl Harbor ferry boat model created for the Pearl Harbor scenario.

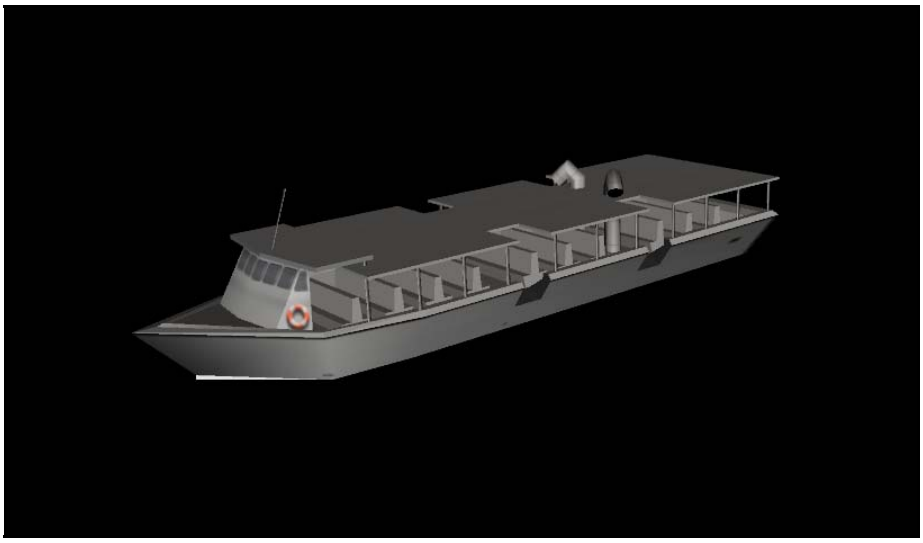


Figure 86. The Arizona ferry boat model created for the Pearl Harbor scenario.

Vizx3D was used as a design tool to identify the locations of piers and navigational aids. Additionally Vizx3D provided the ability to create a visual representation of abstract objects such as the nautical chart. This was performed by taking the simple terrain model for the Pearl Harbor scenario and then creating 3D overlays of environment components of interest. Figure 84 shows a close-up view of a portion of the Pearl Harbor setup scene as viewed in Vizx3D.

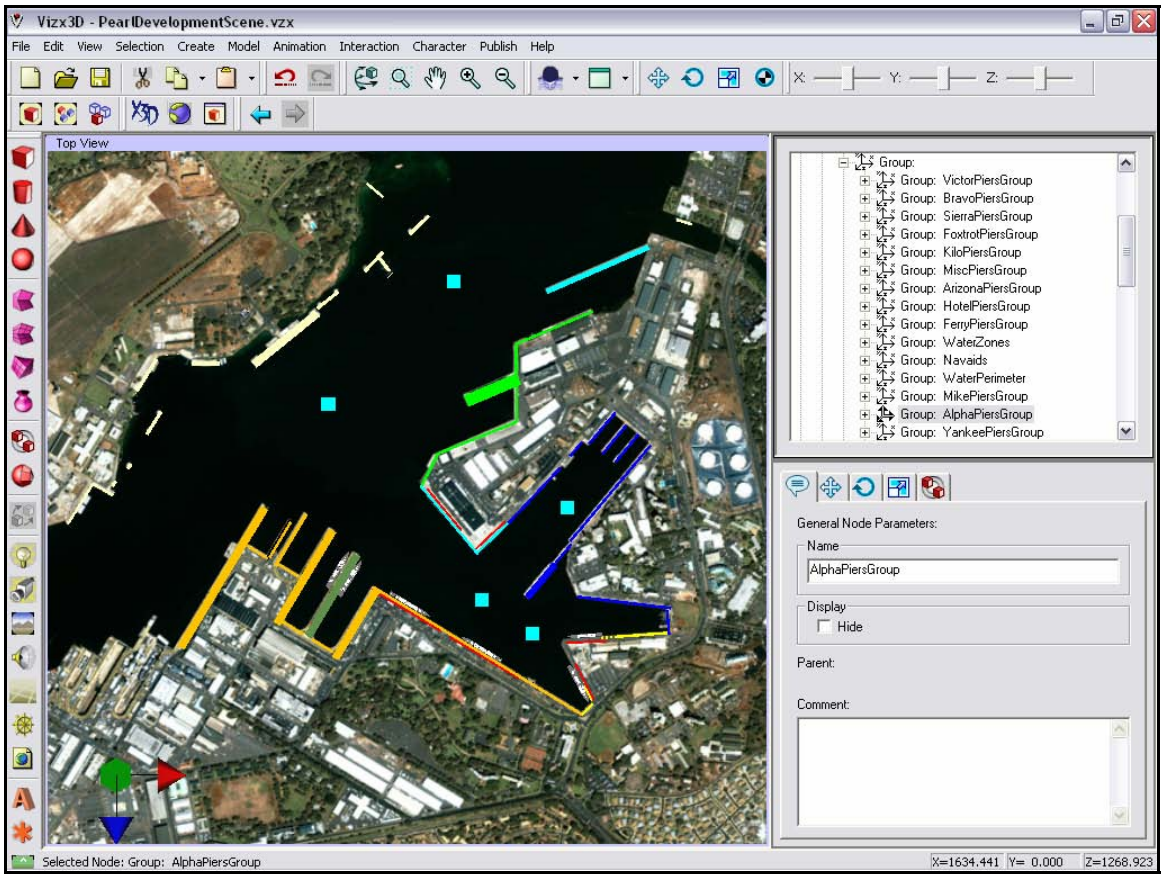


Figure 87. Illustrates how 3D authoring tools were used to create overlays of objects for agent environment design

The scene graph window to the right shows that the overlay objects have been organized and sorted based on the names of the object in the real-world. An additional benefit to this approach is the ability for a scenario author to navigate the entire virtual environment from 3D and 2D or top-down perspectives. This freedom of navigation provides a scenario author with a detailed understanding how the environment may impact the agents in a simulation. This approach also provides scenario authors with a means to explain and discuss simulation design with subject matter experts that may have a better understanding of the actual real-world environment. Figure 85 shows the same view but with the terrain removed.

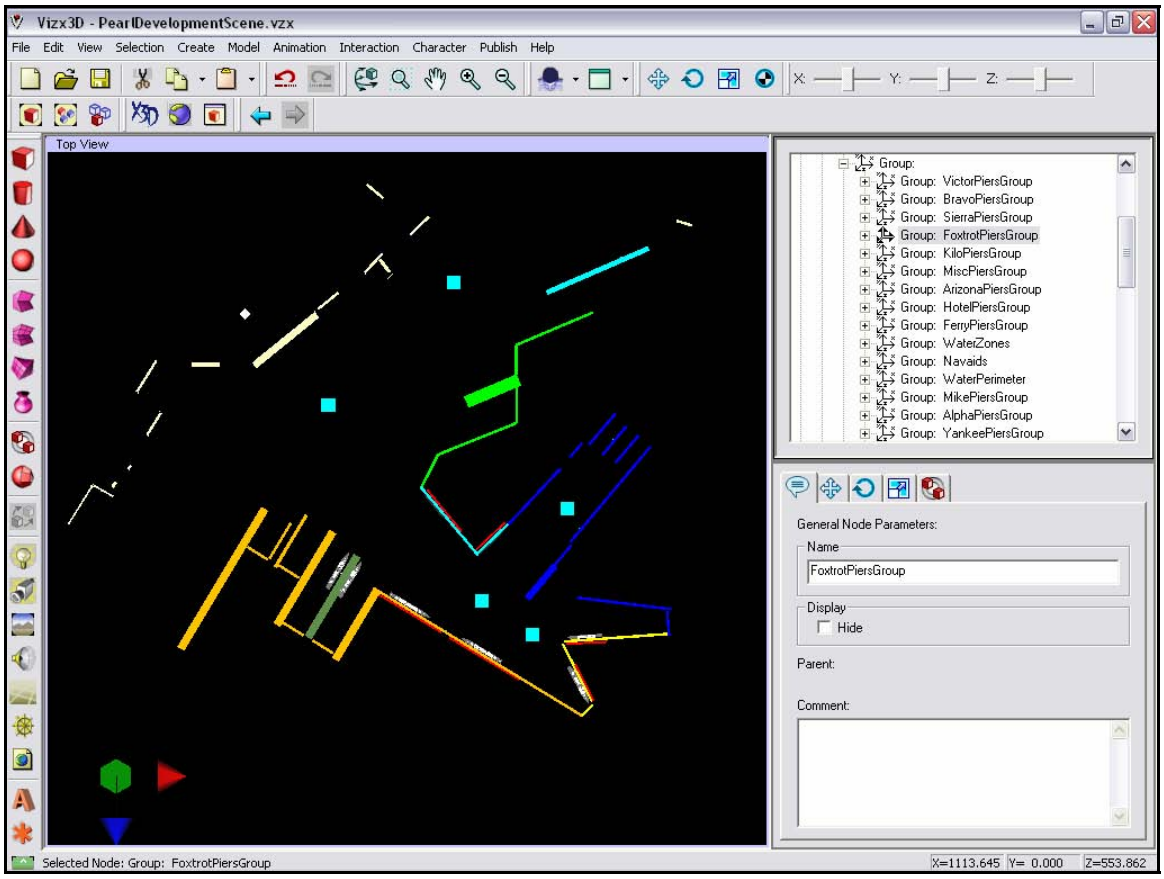


Figure 88. Illustrates the agent environment design of the Pearl Harbor scenario with only the overlay objects visible.

Using the objects in Figure 85, the scenario author can create a DES environment that is representative of the virtual world. In this example, 3D geometry objects were used to identify the length and width of the piers. The same approach was used to identify the length, width, and height of navigational buoys that are present in the harbor. With this information the scenario was authored in Viskit accounting for each of these objects. The ability to create and save a 3D representation of these objects allows an author to test and evaluate the design of the agent environment and identify any potential problems with the implementation. This approach would not have been possible if Planet9 studios did not provide low-fidelity versions of the model early in the development process. It is recommended that future projects of this type use a similar approach to allow for parallel development by all project partners.

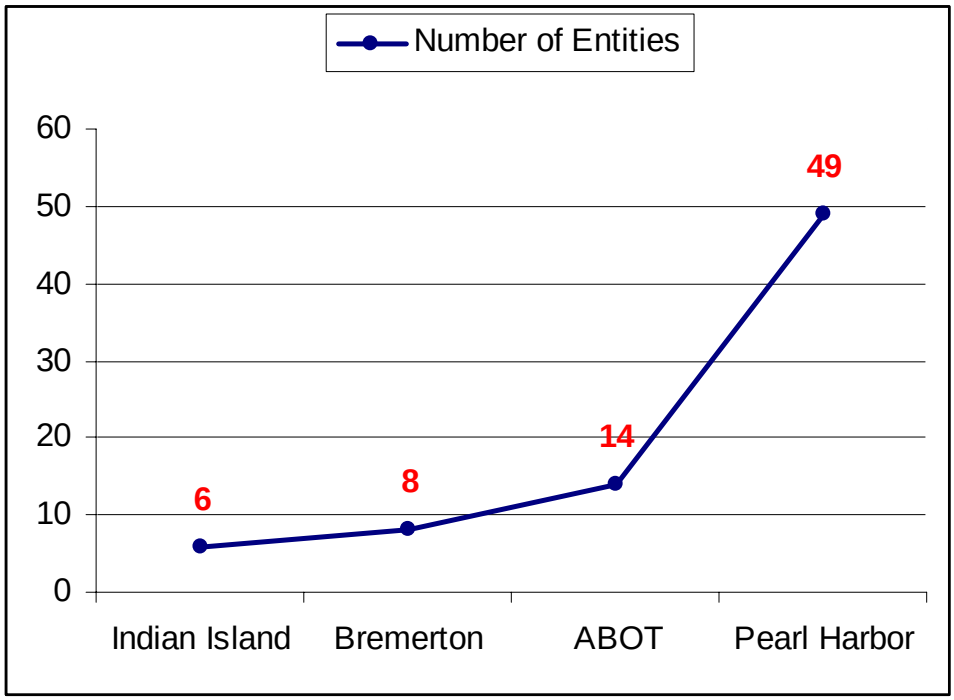


Figure 89. Depicts the relationship between the size of the environment and the number of entities that are used in respective scenarios.

Throughout this project it was apparent that the size of the environment directly impacts the number of entities that might be sensibly used in the simulation. Figure 86 shows the number of entities that were used in each simulation. This is an important point since adding complexity to a scene can impact a 3D browser's ability to display the results. Ultimately performance is dependent on the computer hardware being used and complexity of the scenario modeled. (Harney 2003) discusses the importance of creating M&S solutions that can be used on computer systems commonly available in the fleet. To ensure that this is done testing and evaluation of simulations should be done on hardware similar to that found deployed with military forces. The 3D models prior to Pearl Harbor were less then 50 megabytes in size. In comparison, the total file size for the Pearl Harbor environment was in excess of 400 megabytes. To ensure users with normal computing hardware can use the simulation, a lower-fidelity version of Pearl Harbor was created using smaller texture files. The resultant environment was only 70 megabytes in size and more easily handled by currently commonplace computer hardware.

When developing new locations, it remains sensible to produce the highest fidelity possible. Computer graphics performance continues to grow at an exponential rate, and fast computers can easily be purchase at low cost.

3. Behavior Definitions

The problem definition for the Pearl Harbor scenario identified a number of behaviors that needed to be developed for the simulation. Using the experience from earlier scenarios and the existing behavior library, these event graphs were constructed in approximately fifteen hours of work. Table 7 lists the new event graphs that were created with a brief description of their functionality.

Event Graph	Description
Buoy	An extension of the obstacle event graph. This simple object has dimensions and should be avoided by moving entities.
Pier with Berths	An extension of the pier event graph that includes the exact location of berths on the pier.
Security Boat	An extension of the military patrol craft event graph. This entity uses the nautical chart object to conduct patrols of the entire harbor environment.
Ship's Self Defense Force (SSDF)	A watch stander that mans a weapons station in port for defense against a threat. This entity communicates on a ship radio circuit and requires a command and control entity to perform its duties.
Pier-side ship	An extension of the military ship event graph. This entity assigns watch positions for the SSDF event graph and serves as the command and control element that the SSDF event graph requires.
Harbor Ferry	An extension of the DISMover3D event graph. This entity follows a pre-determined ferry route. This behavior was used for both ferry types in the Pearl Harbor scenario.
Harbor Control Tower	An entity that has a surface radar sensor and communicates both with un-identified contacts and the other friendly force entities in the harbor.

Table 7. List of the event graphs tcreated specifically for the Pearl Harbor scenario.

The majority of the event graphs that were created for Pearl Harbor were modified and/or enhanced versions of existing behaviors. By using existing, proven examples, a number of behaviors could be modeled in a relatively short period of time.

4. Simulation Configuration

After the requisite event graphs were created the Pearl Harbor Scenario was constructed into a Viskit assembly. Since the entire harbor environment was being modeled the number of objects in the assembly was much larger. Figure 87 shows the complete Pearl Harbor assembly file.

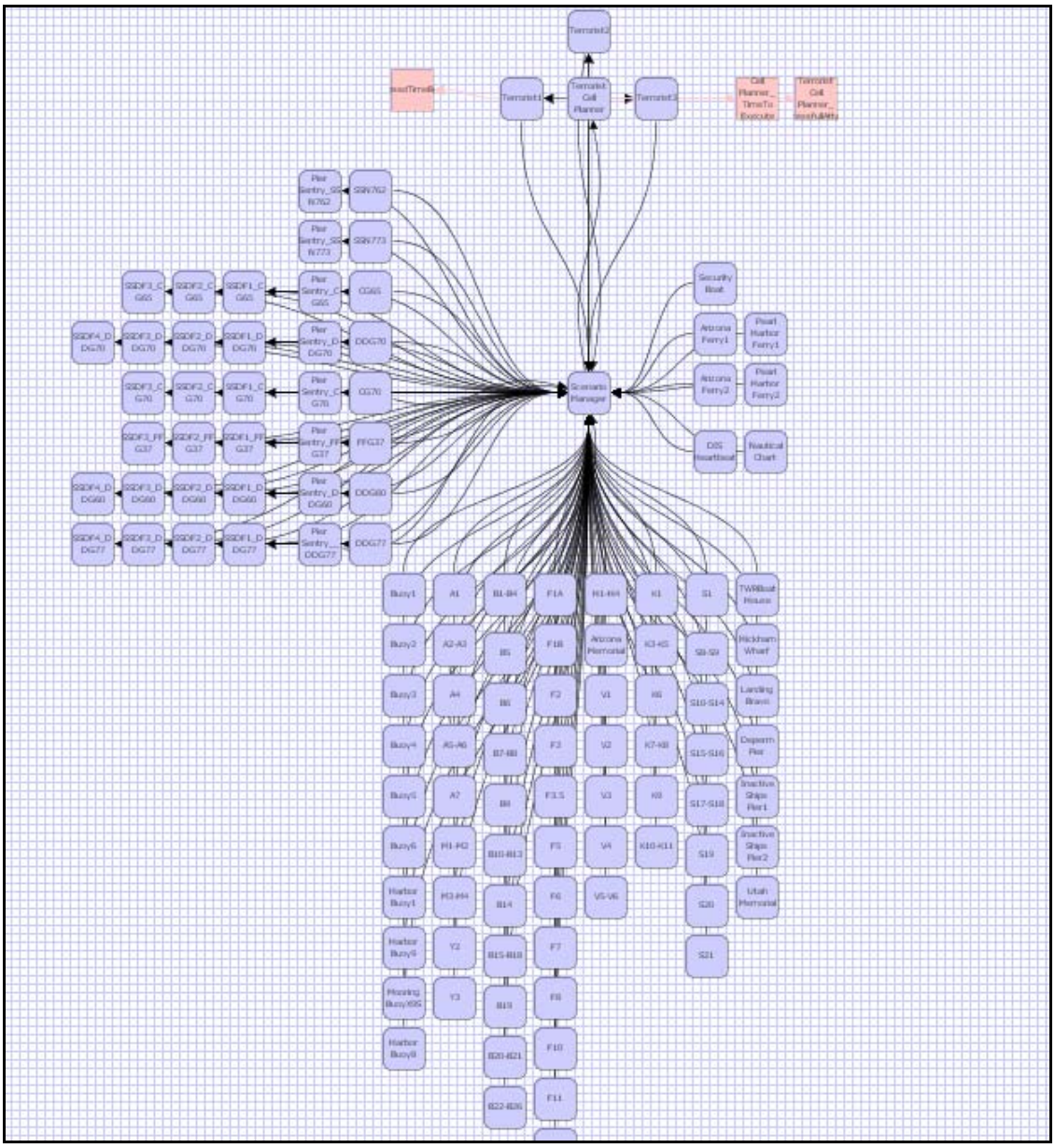


Figure 90. The Pearl Harbor scenario displayed in the Viskit assembly editor panel.

Figure 88 shows a closer view of the bottom grouping of SimEntities in the Pearl Harbor assembly file. This portion of the assembly file was constructed using the 3D object design discussed earlier in this chapter. Following this pattern the majority of the agent environment can be designed while a complete 3D model is being built.

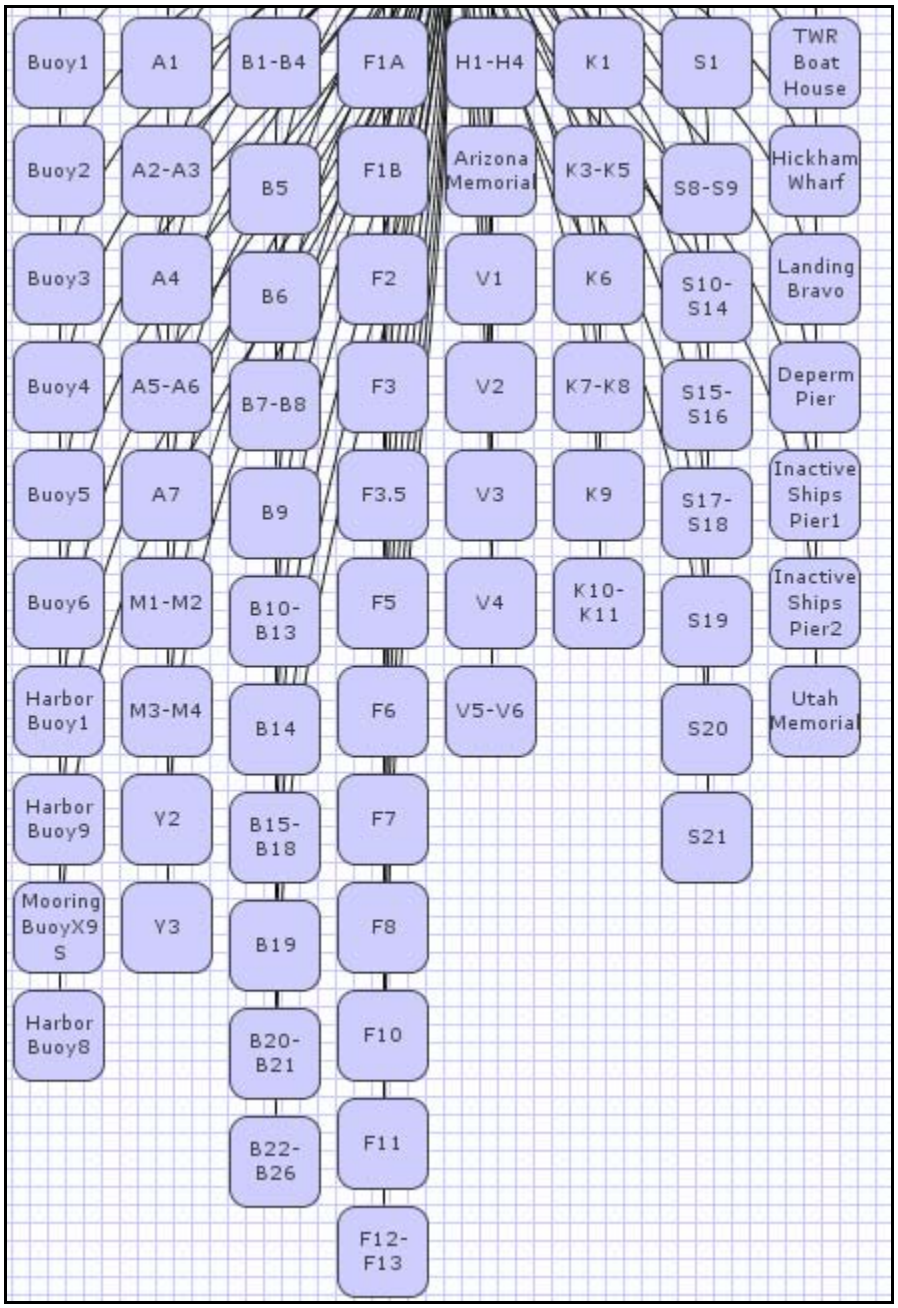


Figure 91. Depicts the environment objects of the Pearl Harbor Scenario assembly file that were created using a 3D authoring tool.

Figure 91 shows that all of the buoy and pier objects discussed earlier in the chapter were included in the assembly file. This graphical representation is another means for subject matter experts to understand the simulated environment. The visual representation of familiar real-world objects can be used to explain an implementation to these experts may not understand the entire simulation design.

Once the final 3D model is finished the 3D layout design created using Vizx3d can be compared to the final harbor model and checked for inconsistencies. In the case of Pearl Harbor, slight variations in pier height were found between the two models. Making the necessary adjustments in the assembly file took less than an hour and allowed for quick integration of the DES model and the 3D visualization.

C. FUTURE ENHANCEMENTS FOR REAL-WORLD STUDIES

This thesis provides a framework and foundation for creating agent-based simulations using DES. The tools used in this project have matured to a level that allows future analysts to add functionality that will allow for real-world studies of AT/FP problems. This section discusses this additional functionality and provides a recommended approach for developing a real-world study.

1. Identifying the Problem

The first step in developing a real-world study is to consult subject matter experts at the installation level to determine the questions that they are trying to answer. Using the guidance from these individuals, future analysts can create a design document outlining the necessary additional functionality and the plan to incorporate required features.

2. Requisite Level of Simulation Expertise

The development process should focus entirely on the real-world simulation and take full advantage of the results from this thesis. This requires that the individuals conducting the study be familiar with DES and proficient in its implementation. The development of user interfaces to automate the simulation design process should not be incorporated into this study. Further development of this type of interface can proceed once the results of the study have been presented to subject matter experts and the simulation design accepted.

3. Optimization of Existing 3D Models

The location of the real-world study can be one of the three locations modeled for this project. Additional work can be devoted to optimizing the existing 3D models and enhancing their performance on hardware commonly found at military commands. These

enhancements include binary compression of the X3D files used, the incorporation of sound, and overlays that enhance the understanding of the environment.

4. Viskit Simulation Enhancements

The additional functionality developed for this study can be designed to answer the specific questions posed by subject matter experts. Chapter IX outlines many recommended enhancements to the existing behavior libraries. As a result of this study a stable release of Viskit can be made available that incorporates the functionality developed during the study.

D. SUMMARY

The Pearl Harbor scenario provided the first opportunity to apply the approaches developed in this thesis to a full scale harbor model. During the development process unique harbor characteristics required the development of additional event graphs and 3D models. The research approach of this thesis was designed to allow for the repeatability and scalability of exemplar scenarios. As a result the creation of these additional objects took a relatively short period of time. Using the development practices of the three early scenarios, a full-scale environment was created by geographically separated development partners in four and a half months. The benefits of parallel development used for Pearl Harbor can be further leveraged for future projects of this size or greater.

IX. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS AND RESULTS

1. Repeatable Framework for Scene Generation

This research has provided a consistent repeatable framework for creating large agent-based simulations, driven by event graph models, with visualized 3D graphics. The tools introduced and discussed in this thesis have greatly reduced the time required for modelers and analysts to create complex environments and behaviors. Interfaces that provide the ability to autogenerate 3D models and DES files allow analysts with a variety of experience and expertise to create and run large-scale simulations. The approach used in this thesis can be used for any simulation problem where an agent-based DES with 3D visualization is desired.

2. Scalable Discrete Event Models with Generated Source Code

The behavior definition library that was created in this thesis provides an example repository that can be used by future users of Viskit for implementing agent-based DES. Behaviors in this library are designed to be expanded and/or altered by future users seeking to build simulations in other problem domains. The behavior definition structure created allows the same entity type (i.e., a Patrol Craft) to be placed in any environment and react accordingly. The autogeneration of source code using Viskit has provided the means to use graphical behavior representations as a base line for implementation understanding. Users are no longer required to be Java programming experts but can now focus on building models at the event-graph level using the behavior library as a reference. This repeatable and scalable approach has provided a foundation for potential future applications, some of which are discussed below.

3. 3D Visualization as an Aid to Event Graph Verification and Development

The use of 3D visualization as a DES development tool was a critical component throughout this work. Using X3D authoring tools like Vizx3D, scenario authors can visualize the agent environment, create visual representations of abstract objects, and implement those components into the DES model. This approach not only assists in the development process but can also be used to explain the underlying framework of agent

environment design. Finally, the ability to view a simulation run during the development process allows for simulation verification of behavior definitions and provides a fundamentally valuable troubleshooting methodology.

B. RECOMMENDATIONS FOR FUTURE WORK

1. Training applications

In addition to analysis applications, this technology is well suited for AT/FP training applications. Creating complex scenarios, representative of the harbor environment, allows students and teachers to interact with and modify the environment. The drag-and-drop functionality of Savage Studio allows users to both develop training scenarios and make modifications to a force protection posture. The impact of these modifications can then be assessed and visualized, greatly enhancing the instruction process. Since the generated source code is written in Java, simple user interfaces can be tied to event-graph models with inputs that can alter the event list and provide man-in-the loop functionality. In this domain a candidate measure of effectiveness (MOE) might be whether or not this combination of technology has improved the student's understanding of specific training objectives for which the scenarios were designed.

2. Increased Sensor Fidelity

The need exists to expand and deepen the sensor libraries of Diskit. The implementation for this thesis was a simple Sphere Cutter Sensor. While this range-based detection model is baseline development and implementation, new sensors need to be created that are representative of the real-world sensors they are designed to simulate. A simple example is the visual perception of a human. The current ability to detect an object is provided by a sensor with 360 degrees of coverage. This sensor should at a minimum be bounded by the field of view of the entity and should include line-of-sight calculations. The level of fidelity required by given real-world scenarios will dictate how this should be modeled.

3. Savage Studio Enhancements

It is recommended that Savage Studio be enhanced to provide the ability for an entire assembly to be authored. Currently a base location is created in Viskit. Entities are then added to the simulation using Savage Studio's drag and drop functionality. It is

recommended that Savage Studio be enhanced to allow complete scene authoring, including scenario development, which can decrease the amount of time required when setting up a harbor environment. Additionally, the Savage Studio interface should be expanded to allow the declaration of property change listeners and unique parameters for a given assembly file. These improvements to Savage Studio will provide a larger group of end users with the ability to create complete scenarios.

4. Real-World Classified Study

This thesis provides an approach and framework that can be applied to a real-world harbor environment. It is recommended that a classified study be conducted in a real-world location to evaluate an actual force protection posture. This study should be collaborative with the force-protection professionals identifying the questions and problems that are addressed by simulation runs and analysis. This study would also be an opportunity to explore the possibility of integrating other resources currently used by force-protection personnel at the chosen installation. Finally, it is logical that the first study should be conducted in a harbor for which a model already exists. Even if the full study is desired in a new harbor, development and testing of the approach can be conducted in a different location while the 3D models for a new location are being developed.

5. Adaptive 'Learning' Terrorist Agents

Currently the terrorist cell planner (TCP) for this project randomly selects combinations of platform, starting position, tactic, and target types for formulating an attack plan. Future work can extend this logic where the TCP uses the success or failure of an attack as part of the planning process. The resultant simulation might enable terrorist agents to optimize their attack plan while holding the defenses of the harbor constant. This approach could provide insight into the greatest vulnerabilities in the harbor-defense strategy.

6. Autogenerated Content from Message Traffic

Both a harbor environment and DES can conceivably be generated using the standard message traffic that is currently used for force protection planning. This follow-on work is likely best modeled utilizing the approach used by (Murray & Quigley 2000) for auto-generating 3D visualizations from air tasking orders (ATOs). Existing

standardized force protection message traffic could be represented in XML which would capture all of the components necessary for auto generating a scenario. This follow-on work would make this application very accessible to the harbor and shipboard FPOs who use this message traffic to specify force protection plans.

7. Benchmark Testing of Cluster Performance

A full study of the use of a high-performance computer cluster as the supplemental computational resource for this application needs to be conducted. Using a set assembly file, benchmark testing should be done to determine when a cluster should be used for running simulations instead of running them locally. Performance data should also be included indicating the correlation between scenario and experiment complexity and the amount of time required to run a simulation.

8. Advanced Behavior Modeling with A* Search

The A* implementation discussed in this thesis should be expanded to incorporate heuristic functions other than simple path optimization. One of the major objectives of force protection is to deter would-be attackers from executing an attack. Future work could apply additional costs to the path-finding heuristic to simulate risk tolerances for other factors such as probability of detection.

9. Implementing Java Inheritance in Viskit XML Based Event Graph Models

Java inheritance was used extensively for defining a structure for agents in this project. This allowed the event graph model to only represent unique and distinguishable aspects of a behavior definition. The parent classes of this structure had to be written in Java because XML based event graphs cannot extend each other. It is recommended that this capability be added to Viskit model representations. If achieved, the event-graph models of need to be created to provide clarity of the inheritance structure that was used.

10. Conducting Risk vs. Cost Assessment

As part of the real-world study parameters could be added to various entities in a simulation which reflect the cost associated with various force protection measures. Analysis can be conducted to compare the overall benefit of implementing measures compared to their cost. This goal requires that cost models for purchases, startup, operation, damage, and replacement be developed behind each tactical entity and shore-side component.

11. Incorporating Measures of Effectiveness (MOEs) and Measures of Performance (MOPs)

When designing future studies, analysts need to utilize accepted MOEs and MOPs to determine what types of questions they are trying to understandably answer. Metron, Inc. a company that was present at the Spring 2006 AT/FP workshop provided a detailed description of widely accepted MOEs and MOPs for AT/FP scenarios. Additional information regarding their presentation can be found at (Brutzman, Blais, & Norbraten 2006).

12. Viskit Documentation

The analyst report output of Viskit needs to be further improved to meet the needs of future analysts. The documentation output developed for this project was an initial example of the ability to generated detailed information about the design and results of simulations. The current output format should be continuously improved as new projects are undertaken. The power of this output mechanism also needs to be enhanced so that DES authors can generate output from Viskit that is not necessarily addressing AT/FP problems.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. ANALYST REPORT INTERFACE AND EXAMPLE GENERATED REPORT

A. ANALYST REPORT INTERFACE

The following eight figures show the Viskit interface that is used for creating an analyst report using Viskit. A brief description is provided for each panel of the interface.

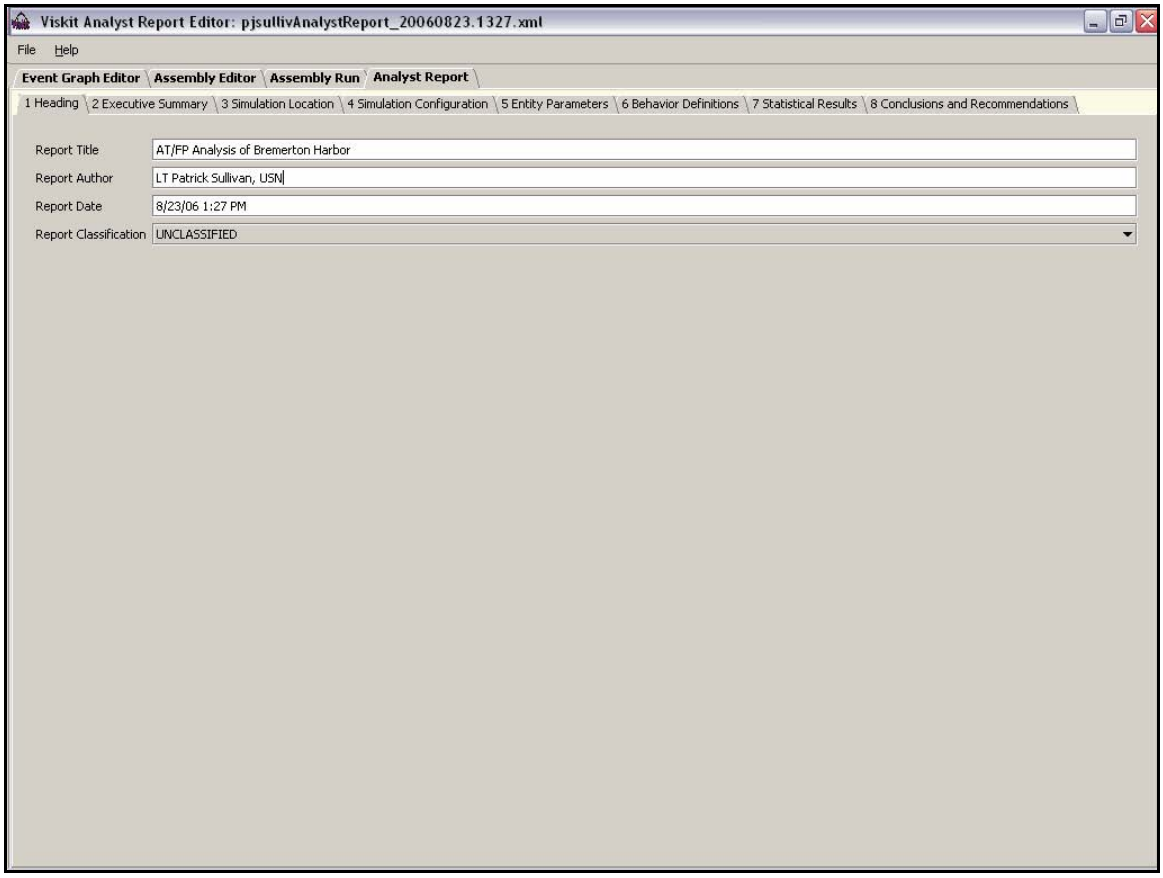


Figure 92. The Heading panel of the Viskit Analyst Report user interface

1. Heading Panel

The heading is used to annotate the title of the report, the author's name, the date and time that the simulation was run, and the classification of the report.

Figure 93. Depicts the Executive Summary panel of the Viskit Analyst Report user interface.

2. Executive Summary Panel

The Executive Summary allows an analyst to provide an overview of the entire simulation, from setup to execution and is listed as the first section of the final report. The analyst comments in this section of the report need to provide a synopsis of the simulation that was conducted including the purpose, design, and overall conclusions. This provides the context for the remainder of the report and ensures that someone reading the report has an overview of the entire simulation problem

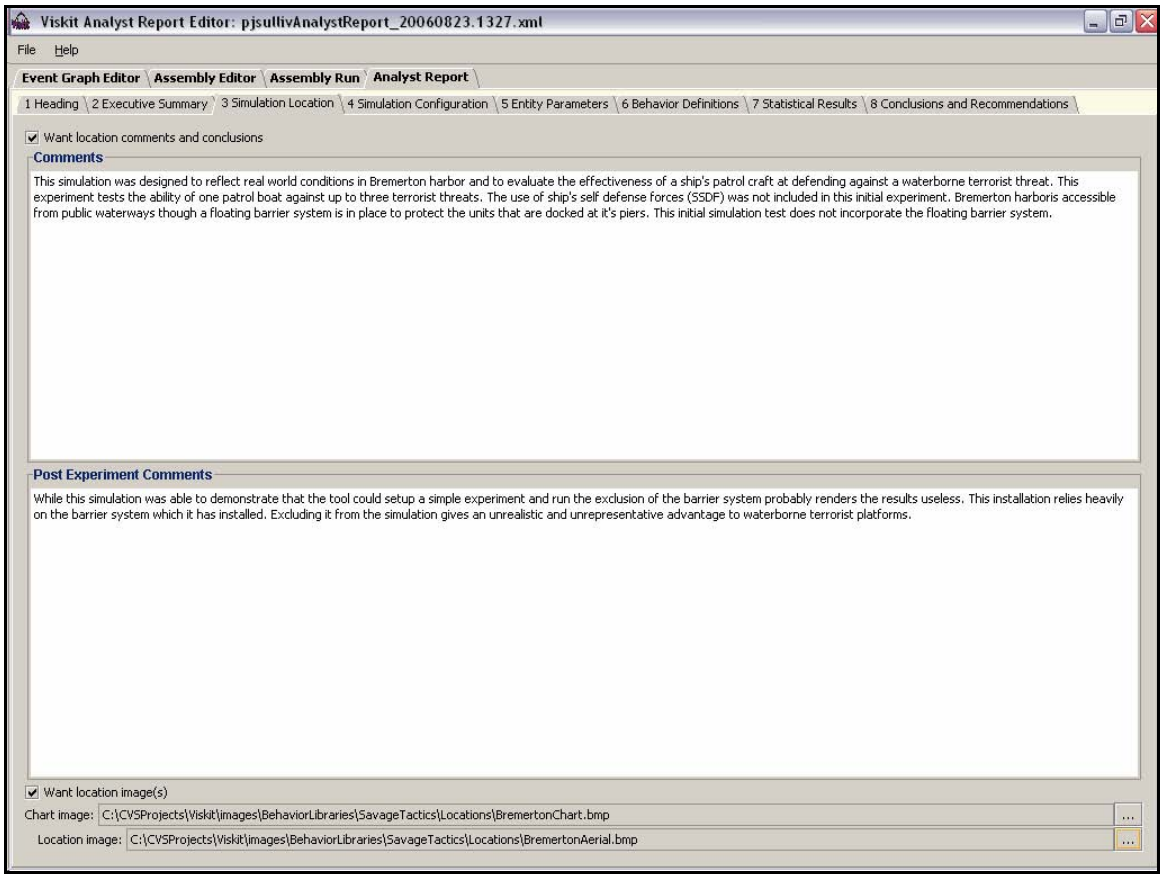


Figure 94. The Simulation Location panel of the Viskit Analyst Report user interface

3. Simulation Location Panel

The Simulation Location panel of the analyst report is used to provide information about the environment of the simulation. Two images can be included in this section and can be specified by the user. The comments in this section allow before and after simulation annotations from an analyst. Analysts need to identify anything about the environment that is not readily apparent to the intended recipient. Using the images in this section the analyst can detail all of the environment considerations and also discuss those elements of the real-world environment that were not included in the simulation.

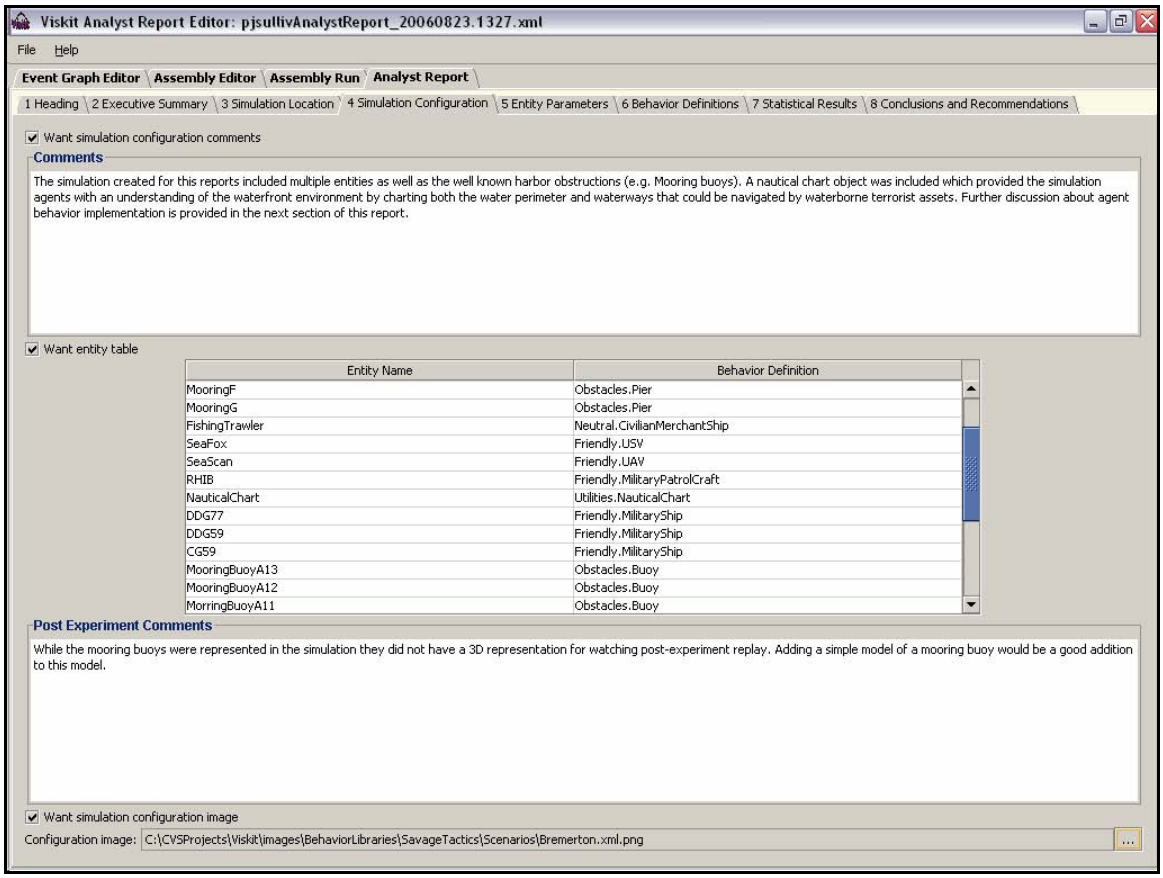


Figure 95. The Simulation Configuration panel of the Viskit Analyst Report user interface

4. Simulation Configuration Panel

The Simulation Configuration panel is used to list all of the entities and objects that were used in the simulation. A table of SimEntities is autogenerated from the assembly file and lists the description of the entity as well as the name of the event graph that defines its behavior. Analysts can provide before and after comments in this section as well. An image of the assembly file can be included in the report from this panel as well. In this section analysts need to provide a detailed explanation of the total configuration. While the image of the assembly serves as an excellent reference for discussion, the analyst must provide as much information as needed to ensure that a reader fully understands the meaning and purpose of the assembly. This information needs to include a discussion about the environment and the connections between entities and their implications.

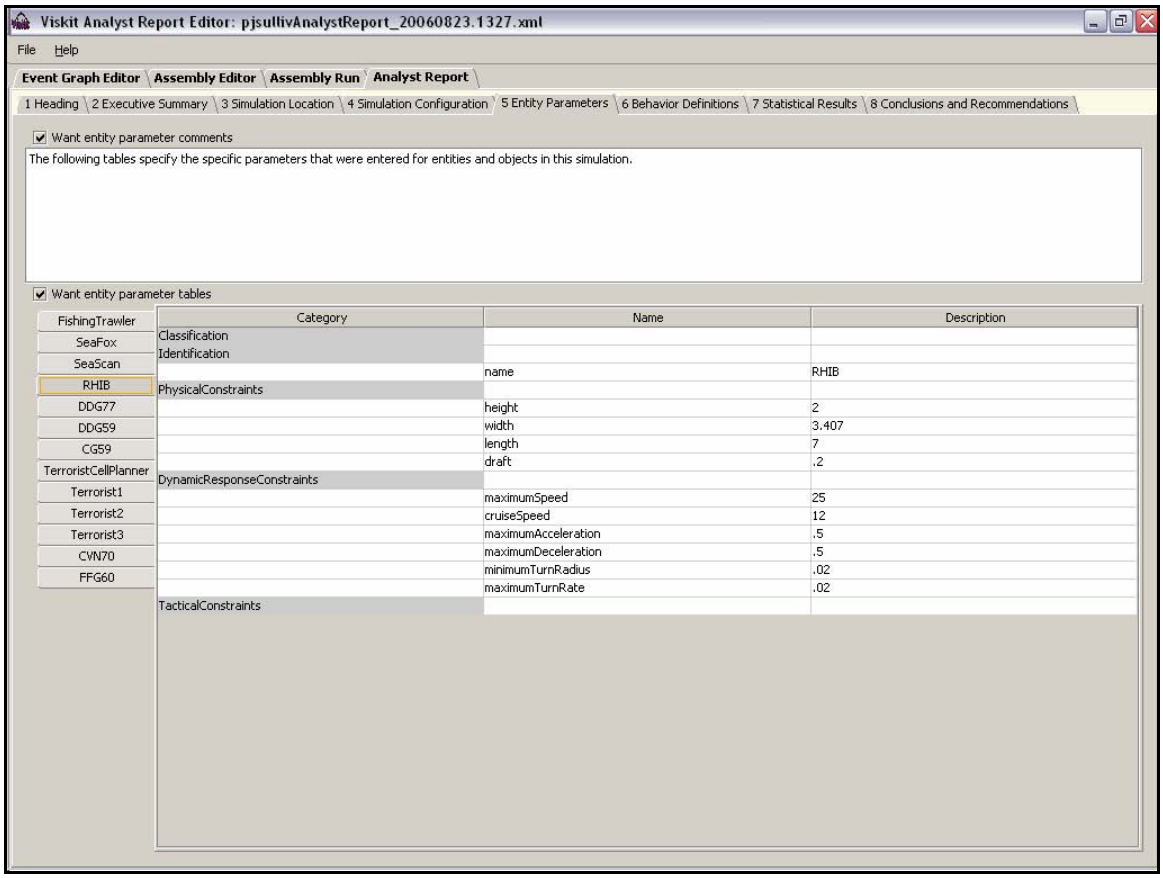


Figure 96. The Entity Parameters panel of the Viskit Analyst Report user interface

5. Entity Parameters Panel

The Entity Parameters panel allows annotation about the parameters used in the simulation and the ability to select whether or not to include tables of all of the parameters that were used. The panel also provides the ability to preview the parameters that will be incorporated. This table is autogenerated by Viskit using the scenario assembly file. Analysts need to provide a discussion about the parameter values that were chosen for the simulation. Analysts should also explicitly point out parameters that were used to influence the outcome of the experiment or answer a specific simulation question.

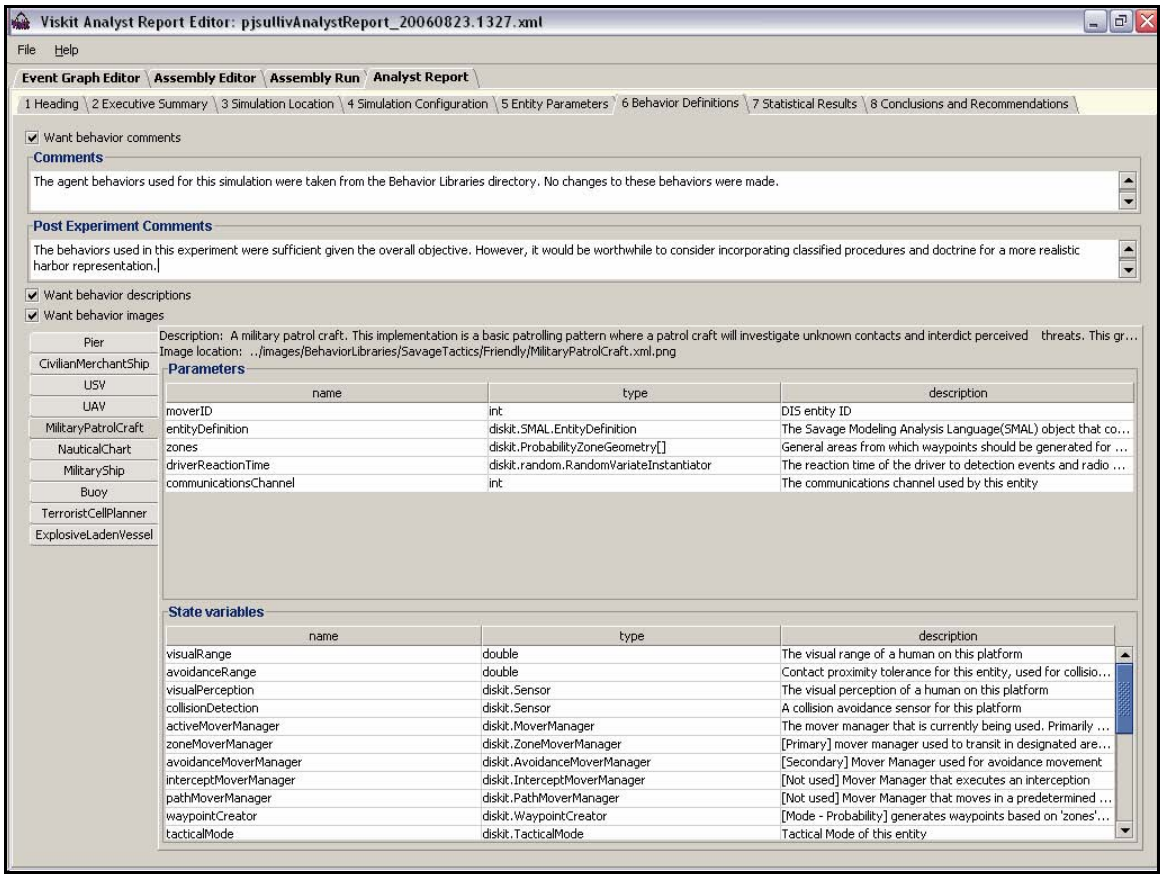
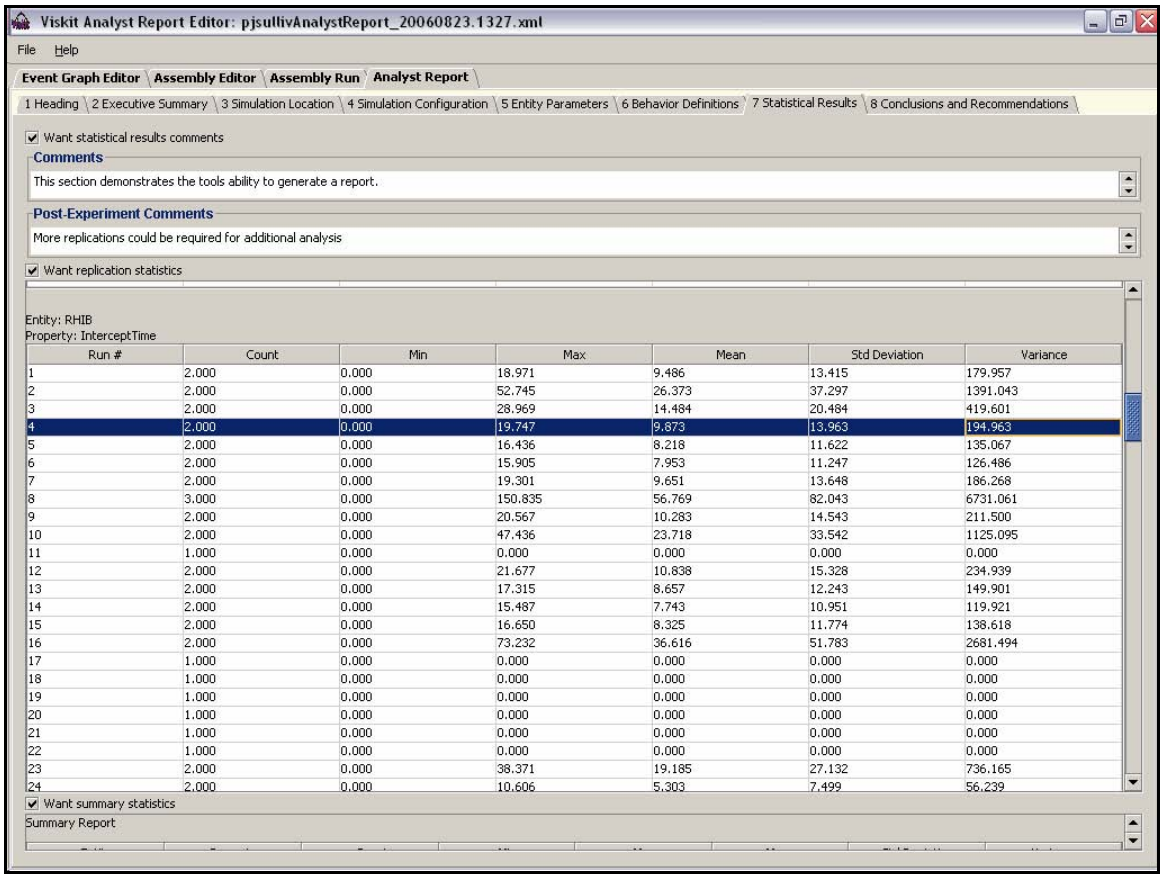


Figure 97. The Behavior Definitions panel of the Viskit Analyst Report user interface

6. Behavior Definitions Panel

The Behavior Definitions panel allows before and after annotation and choices regarding the level of detail for behavior definition descriptions. The behavior definitions portion of the analyst report can provide an image of the event graph as well as a table of all of the parameters and state variables that were used for the simulation. In this panel users can preview the values for the tables prior to generating the report. Analysts need to provide a description about the behaviors used in the simulation. If an analyst believes that simulation artificialities exist in a behavior definition they should explicitly identify those artificialities and provide a discussion about the potential effects on the simulation outcome.



Run #	Count	Min	Max	Mean	Std Deviation	Variance
1	2.000	0.000	18.971	9.486	13.415	179.957
2	2.000	0.000	52.745	26.373	37.297	1391.043
3	2.000	0.000	28.969	14.484	20.484	419.601
4	2.000	0.000	19.747	9.873	13.963	194.963
5	2.000	0.000	16.436	8.218	11.622	135.067
6	2.000	0.000	15.905	7.953	11.247	126.486
7	2.000	0.000	19.301	9.651	13.648	186.268
8	3.000	0.000	150.835	56.769	82.043	6731.061
9	2.000	0.000	20.567	10.283	14.543	211.500
10	2.000	0.000	47.436	23.718	33.542	1125.095
11	1.000	0.000	0.000	0.000	0.000	0.000
12	2.000	0.000	21.677	10.838	15.328	234.939
13	2.000	0.000	17.315	8.657	12.243	149.901
14	2.000	0.000	15.487	7.743	10.951	119.921
15	2.000	0.000	16.650	8.325	11.774	138.618
16	2.000	0.000	73.232	36.616	51.783	2681.494
17	1.000	0.000	0.000	0.000	0.000	0.000
18	1.000	0.000	0.000	0.000	0.000	0.000
19	1.000	0.000	0.000	0.000	0.000	0.000
20	1.000	0.000	0.000	0.000	0.000	0.000
21	1.000	0.000	0.000	0.000	0.000	0.000
22	1.000	0.000	0.000	0.000	0.000	0.000
23	2.000	0.000	38.371	19.185	27.132	736.165
24	2.000	0.000	10.606	5.303	7.499	56.239

Figure 98. The Statistical Results panel of the Viskit Analyst Report user interface

7. Statistical Results Panel

The Statistical Results panel allows a user to provide before and after comments as well as preview the replication and summary statistics for a simulation. A user can also specify whether or not to include histogram charts of the replication statistics as part of their output. In addition to discussing the statistical results the analyst needs to provide a discussion about their analysis of the data and ensure that a reader is not misled by the numerical output.

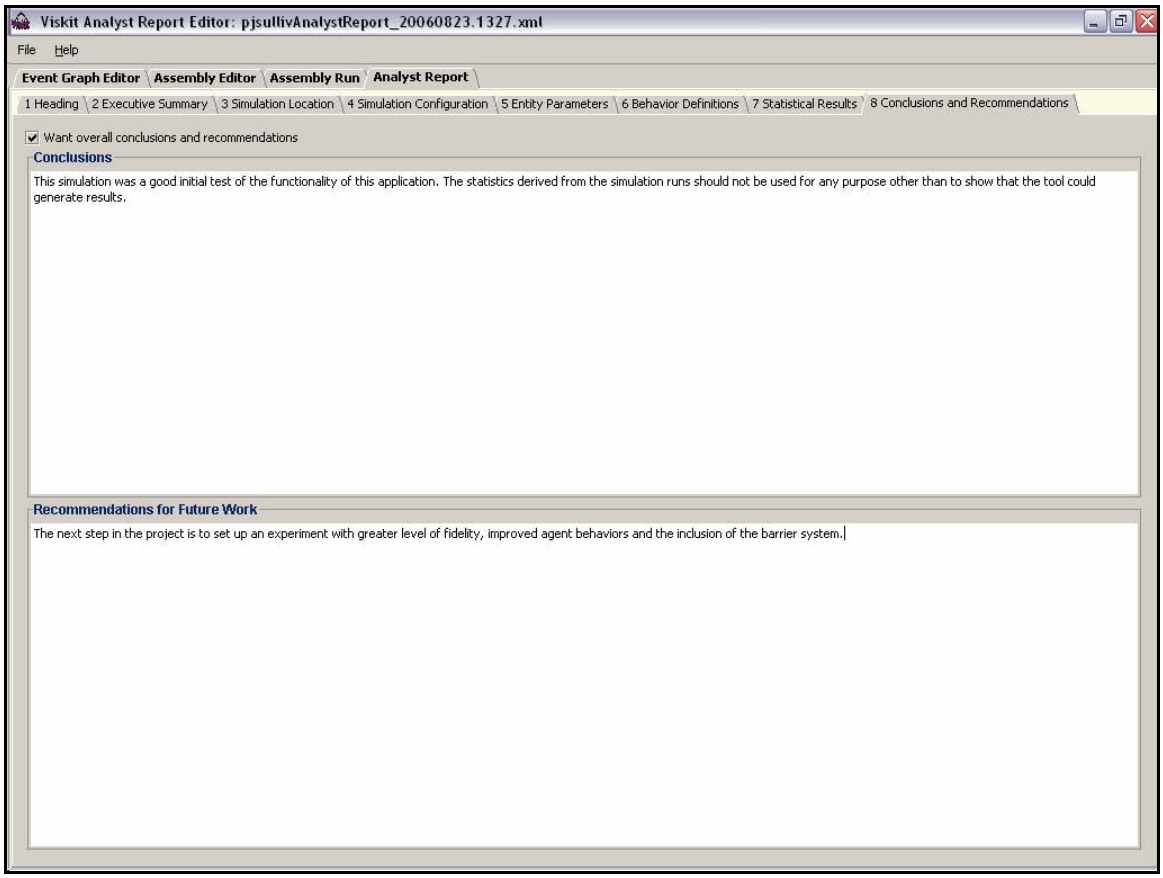


Figure 99. The Conclusions and Recommendations panel of the Viskit Analyst Report user interface

8. Conclusions and Recommendations Panel

The final section of the analyst report interface is the Conclusions and recommendations panel. Analysts can use this interface to provide comments regarding the results of the simulation and recommendations for future work. In this section analysts need to identify the specific questions that were answered. Analysts also need to provide details in their recommendations for future work. These details need to include any required enhancements to simulation design and execution.

B. GENERATED REPORT EXAMPLE

The following example shows portions of the analyst report that was generated using the analyst report interface. The Bremerton scenario was run for fifty replications. The generated report was fifty-one pages in length. For brevity, example excerpts of each section of the report have been included to illustrate the output format.

THIS REPORT IS: **UNCLASSIFIED**

AT/FP Analysis of Bremerton Harbor

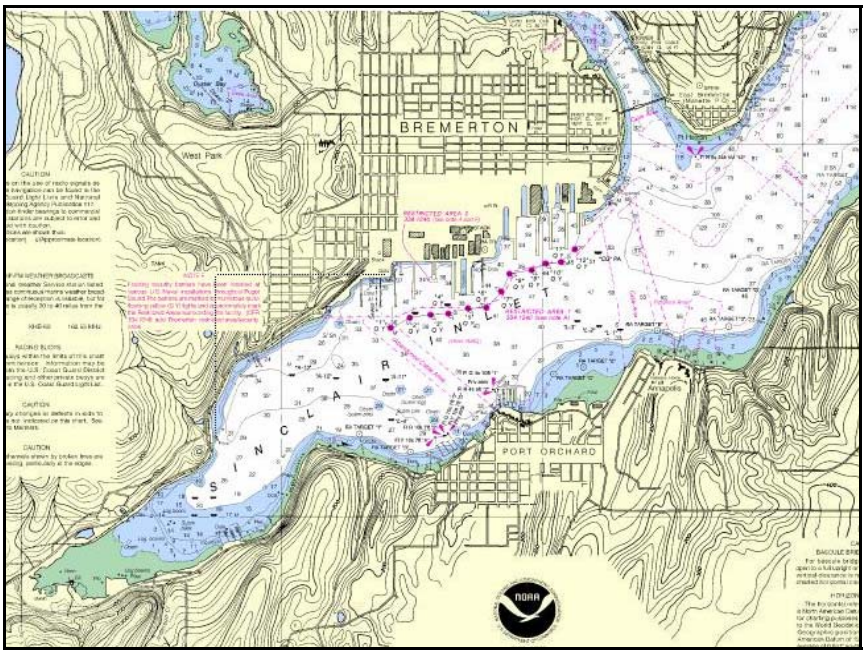
Prepared by: **LT Patrick Sullivan, USN**
Date: **8/23/06 1:27 PM**

Executive Summary
Analyst Executive Summary: The purpose of this report is to test various force protection alternatives for Naval Station Bremerton Washington. The initial motivation for this report is to provide an exemplar template of what the end product of the ATFP tool could be to allow for discussion and suggestions among participating developers and sponsors

---

Simulation Location
Analyst Considerations: This simulation was designed to reflect real-world conditions in Bremerton harbor and to evaluate the effectiveness of a ship's patrol craft at defending against a waterborne terrorist threat. This experiment tests the ability of one patrol boat against up to three terrorist threats. The use of ship's self defense forces (SSDF) was not included in this initial experiment. Bremerton harbor is accessible from public waterways though a floating barrier system is in place to protect the units that are docked at it's piers. This initial simulation test does not incorporate the floating barrier system.

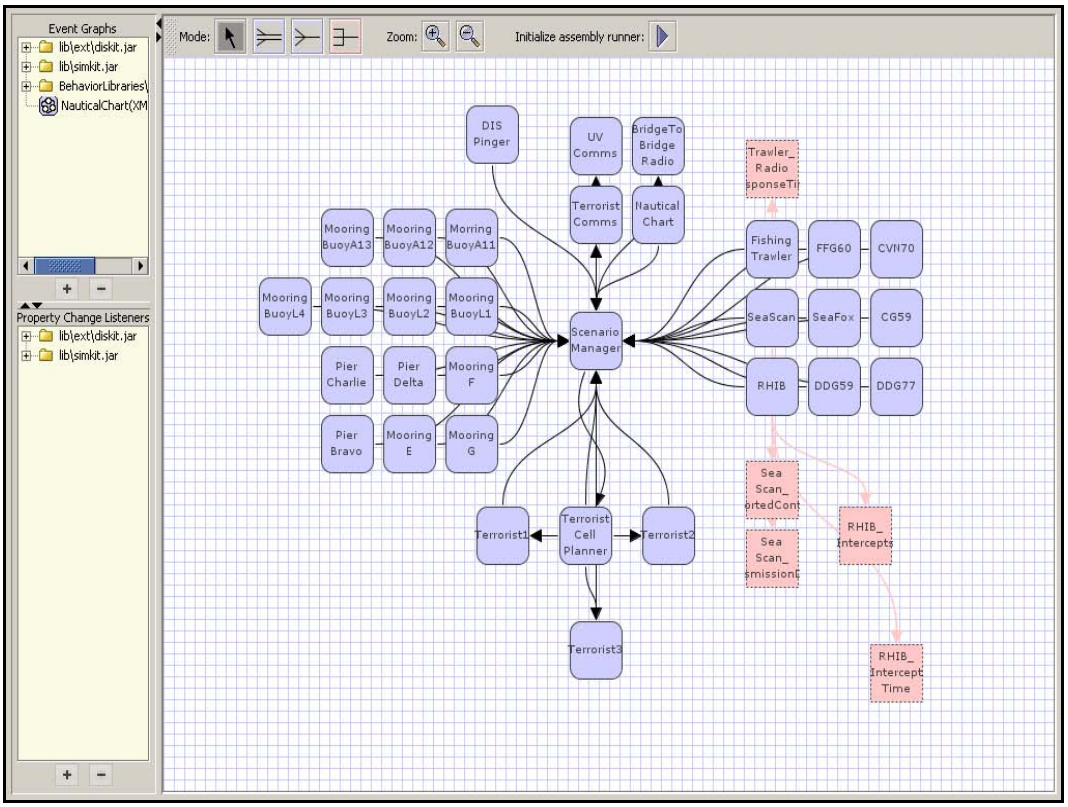
Post-Experiment Analysis: While this simulation was able to demonstrate that the tool could setup a simple experiment and run the exclusion of the barrier system probably renders the results useless. This installation relies heavily on the barrier system which it has installed. Excluding it from the simulation gives an unrealistic and unrepresentative advantage to waterborne terrorist platforms.


BREMERTON
West Park
PORT ORCHARD
SINCLAIR
NOAA

Simulation Configuration

Analyst Discussion: The simulation created for this reports included multiple entities as well as the well known harbor obstructions (e.g. Mooring buoys). A nautical chart object was included which provided the simulation agents with an understanding of the waterfront environment by charting both the water perimeter and waterways that could be navigated by waterborne terrorist assets. Further discussion about agent behavior implementation is provided in the next section of this report.

Post-Experiment Analysis: While the mooring buoys were represented in the simulation they did not have a 3D representation for watching post-experiment replay. Adding a simple model of a mooring buoy would be a good addition to this model.



Simulation Entities

Entity Name	Behavior Definition
PierBravo	Obstacles.Pier
PierCharlie	Obstacles.Pier
PierDelta	Obstacles.Pier
MooringE	Obstacles.Pier
MooringF	Obstacles.Pier
MooringG	Obstacles.Pier
FishingTrawler	Neutral.CivilianMerchantShip
SeaFox	Friendly.USV
SeaScan	Friendly.UAV
RHIB	Friendly.MilitaryPatrolCraft
NauticalChart	Utilities.NauticalChart
DDG77	Friendly.MilitaryShip
DDG59	Friendly.MilitaryShip
CG59	Friendly.MilitaryShip
MooringBuoyA13	Obstacles.Buoy
MooringBuoyA12	Obstacles.Buoy
MorringBuoyA11	Obstacles.Buoy
MooringBuoyL1	Obstacles.Buoy
MooringBuoyL2	Obstacles.Buoy
MooringBuoyL3	Obstacles.Buoy
MooringBuoyL4	Obstacles.Buoy
TerroristCellPlanner	Hostile.TerroristCellPlanner
Terrorist1	Hostile.ExplosiveLadenVessel
Terrorist2	Hostile.ExplosiveLadenVessel
Terrorist3	Hostile.ExplosiveLadenVessel
CVN70	Friendly.MilitaryShip
FFG60	Friendly.MilitaryShip
PierBravo	Obstacles.Pier
PierCharlie	Obstacles.Pier
PierDelta	Obstacles.Pier
MooringE	Obstacles.Pier
MooringF	Obstacles.Pier
MooringG	Obstacles.Pier

Simulation Parameters for: **CG59**

Classification		
	level	UNCLASSIFIED
	reference	http://www.fas.org/man/dod-101/sys/ship/index.html
	rationale	All values in this model are generic estimations of this type of platform
Identification		
	name	USS PRINCETON
Physical Constraints		
	height	57.6
	width	16.764
	length	172.82
	draft	10.058
Dynamic Response Constraints		
	maximumSpeed	30
	cruiseSpeed	15
	maximumAcceleration	5
	maximumDeceleration	5
	minimumTurnRadius	.015
	maximumTurnRate	3
Tactical Constraints		
	maximumAirDetectionRange	74080
	maximumSurfaceDetectionRange	37040
	maximumSubsurfaceDetectionRange	30

Simulation Parameters for: **Terrorist1**

Classification		
	Level	UNCLASSIFIED
	Reference	none
	Rationale	All values in this model are generic estimations of this type of platform
Identification		
	Name	Terrorist 1
Physical Constraints		
	Height	2
	Width	5
	Length	18
	Draft	.5
Dynamic Response Constraints		
	maximumSpeed	25
	cruiseSpeed	18
	maximumAcceleration	5
	maximumDeceleration	5
	minimumTurnRadius	3
	maximumTurnRate	.025
Tactical Constraints		
	maximumSurfaceDetectionRange	10000

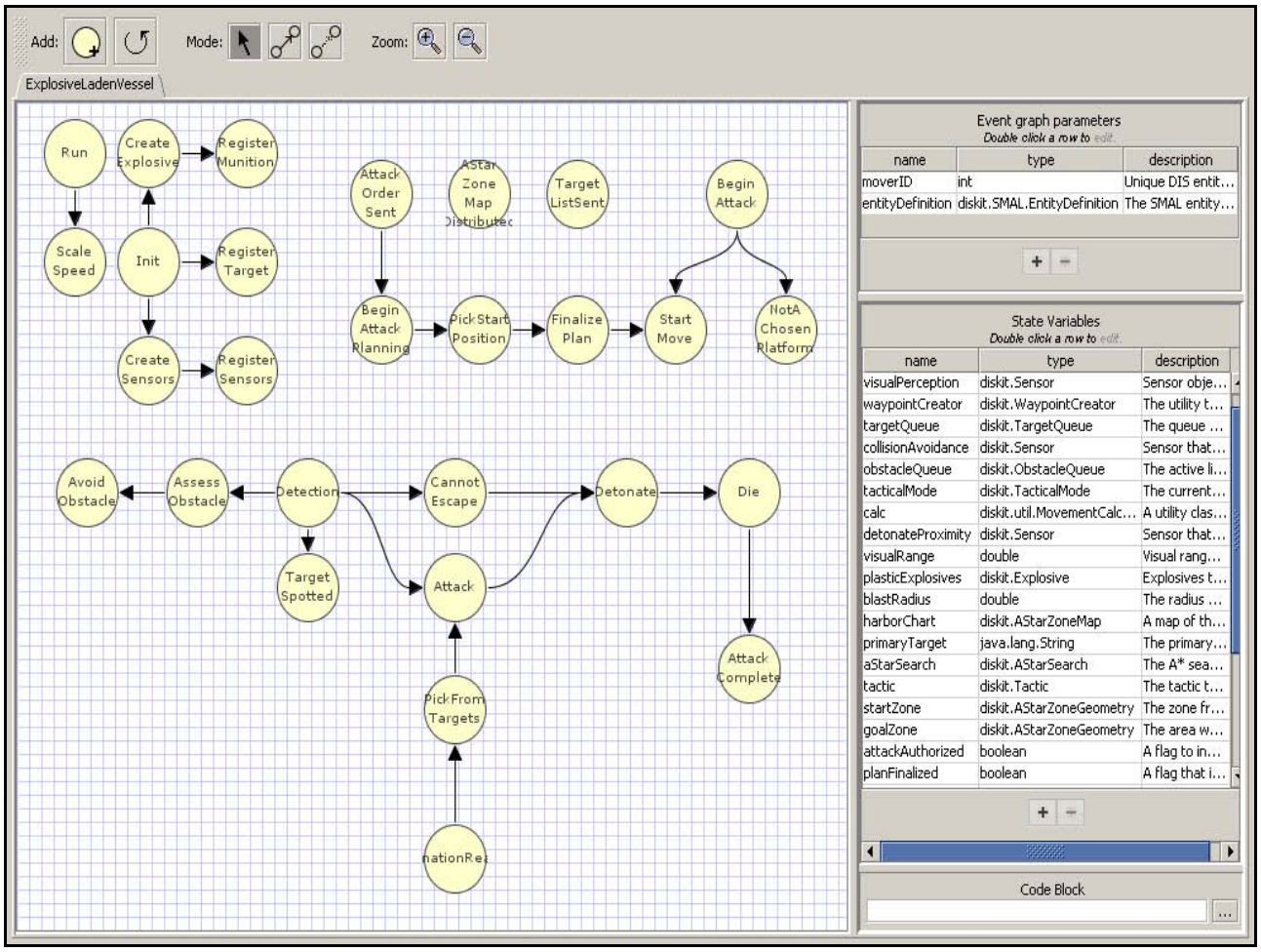
Behavior Definitions

Analyst Discussion: The agent behaviors used for this simulation were taken from the Behavior Libraries directory. No changes to these behaviors were made.

Post-Experiment Analysis: The behaviors used in this experiment were sufficient given the overall objective. However, it would be worthwhile to consider incorporating classified procedures and doctrine for a more realistic harbor representation.

Behavior: ExplosiveLadenVessel

Description: Terrorist that starts from a point randomly within ProbabilityZoneGeometry objects. Terrorist will proceed towards the harbor, scans the area for potential targets and attacks the targets in order of proximity and order in which detected. TODO: Add booleans to the constructor for attack Friendly, attack HVU



Parameter	Parameter Type	Description
moverID	int	Unique DIS entity ID number
entityDefinition	diskit.SMAL.EntityDefinition	The SMAL entity definition for this entity

State Variable	Variable Type	Description
avoidanceRange	Double	The proximity tollerance for this entity
visualPerception	diskit.Sensor	Sensor object that processes all targets in the state space before attacking
waypointCreator	diskit.WaypointCreator	The utility that creates waypoints based on the arguments passed
targetQueue	diskit.TargetQueue	The queue of active targets for this entity
collisionAvoidance	diskit.Sensor	Sensor that implements collision avoidance
obstacleQueue	diskit.ObstacleQueue	The active list of obstacles that this entity is concerned with. Special class that organizes the queue based on the proximity of the obstacles
tacticalMode	diskit.TacticalMode	The current tactical mode of this entity
calc	diskit.util.MovementCalculator	A utility class that performs 3D vector math calculations
detonateProximity	diskit.Sensor	Sensor that determines if the contact is close enough to kill
visualRange	Double	Visual range for this entity
plasticExplosives	diskit.Explosive	Explosives that this entity is carrying
blastRadius	Double	The radius of the explosives
harborChart	diskit.AStarZoneMap	A map of the environment that contains the collection of search nodes for path finding
primaryTarget	java.lang.String	The primary target for this entity
aStarSearch	diskit.AStarSearch	The A* search implementation that is used for path finding
tactic	diskit.Tactic	The tactic that has been ordered for this terrorist
startZone	diskit.AStarZoneGeometry	The zone from which the terrorist should start
goalZone	diskit.AStarZoneGeometry	The area where this terrorist should head towards to execute it's tactics
attackAuthorized	Boolean	A flag to indicate whether or not the cell planner has authorized the execution of the attack plan
planFinalized	boolean	A flag that indicates that this entity has processed it's attack order and is ready to attack

zoneMoverManager	diskit.ZoneMoverManager	The primary mover for this entity, moves using A* search zone geometry
attackStartWaypoint	diskit.Vec3d	The waypoint selected to start the attack
chosenToAttack	boolean	Flag that is set to true when this entity has been selected to attack
arrivedAtGoal	boolean	Flags when the entity has reached it's destination
speedScalar	double	The default speed scale for this entity
attackSuccess	java.lang.String	String report of the result of the attack
destroyedTarget	diskit.Mover3D	The mover that was destroyed
terroristLeader	simkit.SimEntity	The entity that is giving the orders to this terrorist
attackDelay	double	The amount of time from when an order is received
timeUndetected	double	The amount of time from simulation start before this attacker was detected by a friendly force
commsChannel	int	The radio channel for this entity
radioMsg	diskit.RadioCommunication	The communication object that this entity uses to send messages
evasiveManuever	boolean	Whether this entity is performing evasive manuevers

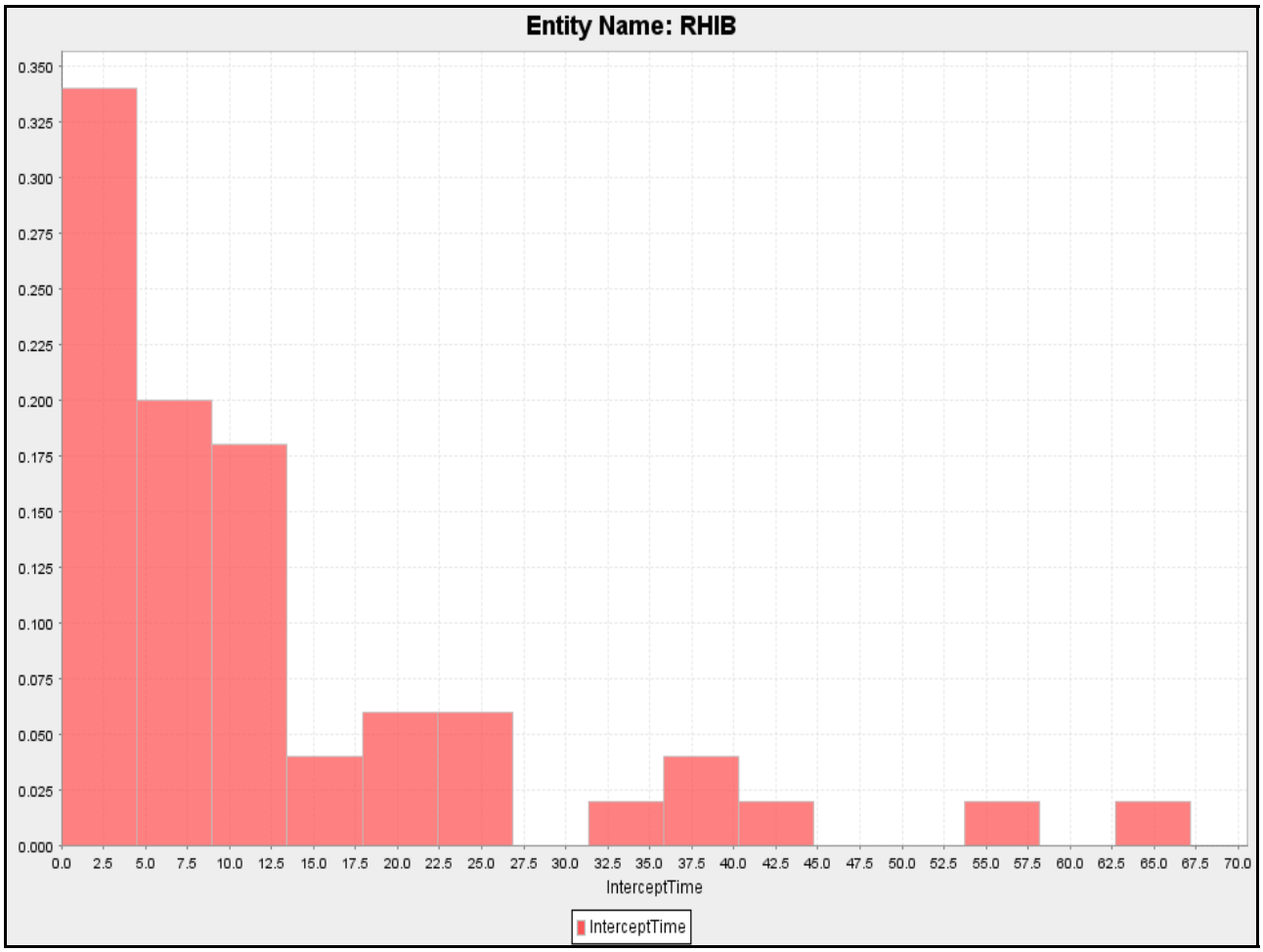
Statistical Results
Analyst Discussion: This section demonstrates the tools ability to generate a report.

Post-Experiment Analysis: More replications could be required for additional analysis

Replication Report :

Entity: **RHIB**

Property: **InterceptTime**



Run#	Count	Min	Max	Mean	StdDev	Variance
1	2.000	0.000	18.971	9.486	13.415	179.957
2	2.000	0.000	52.745	26.373	37.297	1391.043
3	2.000	0.000	28.969	14.484	20.484	419.601
4	2.000	0.000	19.747	9.873	13.963	194.963

5	2.000	0.000	16.436	8.218	11.622	135.067
6	2.000	0.000	15.905	7.953	11.247	126.486
7	2.000	0.000	19.301	9.651	13.648	186.268
8	3.000	0.000	150.835	56.769	82.043	6731.061
9	2.000	0.000	20.567	10.283	14.543	211.500
10	2.000	0.000	47.436	23.718	33.542	1125.095
11	1.000	0.000	0.000	0.000	0.000	0.000
12	2.000	0.000	21.677	10.838	15.328	234.939
13	2.000	0.000	17.315	8.657	12.243	149.901
14	2.000	0.000	15.487	7.743	10.951	119.921
15	2.000	0.000	16.650	8.325	11.774	138.618
16	2.000	0.000	73.232	36.616	51.783	2681.494
17	1.000	0.000	0.000	0.000	0.000	0.000
18	1.000	0.000	0.000	0.000	0.000	0.000
19	1.000	0.000	0.000	0.000	0.000	0.000
20	1.000	0.000	0.000	0.000	0.000	0.000
21	1.000	0.000	0.000	0.000	0.000	0.000
22	1.000	0.000	0.000	0.000	0.000	0.000
23	2.000	0.000	38.371	19.185	27.132	736.165
24	2.000	0.000	10.606	5.303	7.499	56.239
25	2.000	0.000	78.265	39.132	55.341	3062.676
26	2.000	0.000	19.588	9.794	13.851	191.847
27	1.000	0.000	0.000	0.000	0.000	0.000
28	2.000	0.000	34.395	17.197	24.321	591.500
29	1.000	0.000	0.000	0.000	0.000	0.000
30	1.000	0.000	0.000	0.000	0.000	0.000
31	2.000	0.000	134.279	67.139	94.949	9015.398
32	1.000	0.000	0.000	0.000	0.000	0.000
33	2.000	0.000	22.666	11.333	16.027	256.863
34	1.000	0.000	0.000	0.000	0.000	0.000
35	2.000	0.000	16.525	8.262	11.685	136.534
36	2.000	0.000	19.048	9.524	13.469	181.417
37	2.000	0.000	15.877	7.938	11.227	126.037
38	1.000	0.000	0.000	0.000	0.000	0.000
39	2.000	0.000	17.112	8.556	12.100	146.418

40	2.000	0.000	37.743	18.871	26.688	712.264
41	1.000	0.000	0.000	0.000	0.000	0.000
42	2.000	0.000	70.772	35.386	50.043	2504.309
43	1.000	0.000	0.000	0.000	0.000	0.000
44	1.000	0.000	0.000	0.000	0.000	0.000
45	2.000	0.000	16.576	8.288	11.721	137.383
46	2.000	0.000	36.465	18.232	25.785	664.848
47	1.000	0.000	0.000	0.000	0.000	0.000
48	2.000	0.000	86.695	43.348	61.303	3758.052
49	2.000	0.000	23.453	11.726	16.584	275.018
50	2.000	0.000	50.257	25.129	35.537	1262.893

Summary Report :

Entity	Property	Count	Min	Max	Mean	StdDev	Variance
Trawler	RadioResponseTime	50.000	0.000	43.345	7.983	16.144	260.632
RHIB	Intercepts	50.000	0.000	33.000	10.820	11.491	132.038
RHIB	InterceptTime	50.000	0.000	67.139	12.267	15.155	229.684
SeaScan	ReportedContacts	50.000	0.000	58.000	18.890	19.267	371.207
SeaScan	TransmissionDelay	50.000	0.000	5.514	3.609	2.279	5.192

Conclusions and Recommendations

Conclusions This simulation was a good initial test of the functionality of this application. The statistics derived from the simulation runs should not be used for any purpose other than to show that the tool could generate results.

Recommendations for future work :The next step in the project is to set up an experiment with greater level of fidelity, improved agent behaviors and the inclusion of the barrier system.

APPENDIX B. APPLYING XSLT IN THE JAVA PROGRAMMING LANGUAGE

A. INTRODUCTION

In Chapter V we discussed how XSLT was used to transform an Analyst Report XML document into an HTML file. This appendix will discuss how Viskit applied the XSLT style sheet to the Analyst Report XML file.

B. JAVA AND XSLT

The two things required to perform an XSLT transformation are an XML document and an XSLT Stylesheet. Using the XML transformation library of Java a simple class can be written that will apply a Stylesheet and save the resultant XML. In the example below the only requirements are the directory location of the analyst report document, the directory location of the XSLT document, and the file name and location to save the new XML document. This small utility class is all that was required to apply the XSLT transformation.

1. Java Source Code Example for Applying XSLT

```
 1 /*
 2 * XsltUtility.java
 3 *
 4 * Created on March 11, 2004, 4:55 PM
 5 *
 6 * This class was written by CDR Duane Davis for work on the AUV Workbench.
 7 * It was copied to this application to perform XSLT conversions.
 8 *
 9 *@author Duane Davis
10 *@version $Id: XsltUtility.java,v 1.1 2006/08/03 15:41:13 pjsulliv Exp $
11 */
12
13 package viskit.xsd.assembly;
14
15
16
17 import javax.xml.transform.*;
18 import javax.xml.transform.stream.StreamResult;
19 import javax.xml.transform.stream.StreamSource;
20 import java.io.FileInputStream;
21 import java.io.FileNotFoundException;
22 import java.io.FileOutputStream;
23
24
25 public class XsltUtility
26 {
27
28     /** Creates a new instance of XmlUtilities */
29     public XsltUtility() {}
30
31
32     /**
33      * Runs an XSL Transformation on an XML file and writes the result to another file
```

```
34      *
35      * @param inFile XML file to be transformed
36      * @param outFile output file for transformation results
37      * @param xslFile XSLT to utilize for transformation
38      *
39      * @return the resulting transformed XML file
40      */
41     public static boolean runXslt(String inFile, String outFile, String xslFile)
42     {
43         try // FileNotFoundException, TransformerConfigurationException,
                                          TransformerException
44         {
45             TransformerFactory factory = TransformerFactory.newInstance();
46             Templates template  = factory.newTemplates(new StreamSource(new
                   FileInputStream(xslFile)));
47             Transformer xFormer = template.newTransformer();
48             Source source = new StreamSource(new FileInputStream(inFile));
49             Result result = new StreamResult(new FileOutputStream(outFile));
50             xFormer.transform(source, result);
51         } // try
52         catch (FileNotFoundException e)
53         {
54             System.out.println("Unable to load file for XSL Transformation\n" +
55                                "   Input file : " + inFile + "\n" +
56                                "   Output file: " + outFile + "\n" +
57                                "   XSLT file  : " + xslFile);
58             return false;
59         } // catch (FileNotFoundException e)
60         catch (TransformerConfigurationException e)
61         {
62             System.out.println("Unable to configure transformer for XSL
                                                        Transformation");
63             return false;
64         } // catch (TransformerConfigurationException e)
65         catch (TransformerException e)
66         {
67             System.out.println("Exception during XSL Transformation");
68             return false;
69         } // catch (TransformerException e)
70
71         return true;
72     } // runXslt
73
74 } // XmlUtilities
```

APPENDIX C. JFREECHART APPLICATION

A. INTRODUCTION

This appendix will show the Java code implementation for using the JFreeChart API to create histograms. The ability to create charts was desired for the analyst report so that graphical representations of the statistics could be realized.

B. JFREECHART EXAMPLE — GENERATING HISTOGRAMS

Using JFreeChart is a relatively straightforward process. The ChartDrawer class of Viskit is responsible for taking output data from a simulation and generating a chart representation. The code for this class is provided below as an example of using the JFreeChart package.

```
 1 /*
 2  * ChartDrawer.java
 3  *
 4  * Created on August 3, 2006, 10:21 AM
 5  *
 6  * This class creates chart objects using the JFreeChart package.
 7  *
 8  *@author Patrick Sullivan
 9  *@version $Id: ChartDrawer.java,v 1.3 2006/08/04 23:31:17 pjsulliv Exp $
10  */
11
12 package viskit.xsd.assembly;
13
14 import org.jfree.chart.ChartFactory;
15 import org.jfree.chart.ChartPanel;
16 import org.jfree.chart.JFreeChart;
17 import org.jfree.chart.plot.PlotOrientation;
18 import org.jfree.data.xy.IntervalXYDataset;
19 import org.jfree.data.statistics.HistogramDataset;
20 import org.jfree.data.statistics.HistogramType;
21 import java.io.OutputStream;
22 import org.jfree.chart.ChartUtilities;
23 import java.io.File;
24 import java.io.FileOutputStream;
25 import java.io.*;
26
```

```
27 public class ChartDrawer {
28
29     private String url;
30     /** Creates a new instance of ChartDrawer */
31     public ChartDrawer() {
32     }
33
34     /**
35      * Creates a histogram image in PNG format based on the parameters provided.
36      *
37      *@param data an array of doubles that are to be plotted
38      *@param outFileName the name of the file to save the image out to
39      *@return url the path name of the created object
40      */
41     public String createHistogram(String title, String label, double[] data, String
                                   fileName){
42         String fileLocation = "./AnalystReports/charts/"+fileName+".png";
43         String url          = "./charts/"+fileName+".png";
44
45     IntervalXYDataset dataset = createIntervalXYDataset(label, data);
46
47     try{
48           saveChart(createChart(dataset, title, label), fileLocation);
49         }catch (java.io.IOException e) {
50           System.err.println("Unable to create chart image: " + e.getMessage());
51           e.printStackTrace();
52     }
53         return url;
54     }
```

Lines 41 - 54 make up the public method that creates and saves a JFreeChart object. Users of this method provide a chart title, a label for data collected, an array of data points to be plotted and a file name for saving the chart. Line 45 takes the data array and forms an IntervalXYDataset. This dataset is a JFreeChart defined object that is used to create histogram charts. The method call to createChart in line 48 actually creates a chart object.

```
55      /**
56       * Creates a data set that is used for making the histogram
57       */
58      private IntervalXYDataset createIntervalXYDataset(String label, double[] data) {
59
60          HistogramDataset dataset = new HistogramDataset();
61          dataset.setType(HistogramType.RELATIVE_FREQUENCY);
62          dataset.addSeries(label, data, 15);
63
64          return dataset;
65      }
```

The method shown from lines 58-65 creates a special instance of an IntervalXYDataset. In this example a HistogramDataset is created. Additionally, a HistogramType is identified and the data is added to the dataset with a label and number of histogram bins.

```
66      /**
67       * Creates the histogram chart
68       */
69        private JFreeChart createChart(IntervalXYDataset dataset, String title, String
                                        label) {
70          final JFreeChart chart = ChartFactory.createHistogram(
71              title,
72              label,
73              "",
74              dataset,
75              PlotOrientation.VERTICAL,
76              true,
77              false,
78              false
79          );
80
81          chart.getXYPlot().setForegroundAlpha(0.75f);
82          return chart;
83        }
```

The method shown from lines 69-82 create the chart object using JFreeChart's chart factory.

```
 84     /**
 85      *Saves a chart to PNG format
 86      *
 87      */
 88     private void saveChart(JFreeChart chart, String path)throws
                              FileNotFoundException, IOException{
 89
 90
 91         File outFile = new File(path);
 92
 93
 94
 95         FileOutputStream  fos = new FileOutputStream(outFile);
 96         ChartUtilities.saveChartAsPNG(outFile, chart, 969, 641);
 97         fos.close();
 98
 99     }
100
101 }
102
```

The method shown in lines 88-97 saves the chart object in portable network graphics format (PNG) using JFreeChart chart utilities. In this example an output file name is provided, as well as the chart and the dimensions of the chart.

C. SUMMARY

JFreeChart is a powerful tool for generating professional quality charts. The simple histogram used for this project is one of many chart types that are available using the JFreeChart API.

APPENDIX D. SAVAGE MODEL ARCHIVES

A. INTRODUCTION

The SAVAGE model archive contains over one thousand 3D models that are open source and available online.

B. SAVAGE OPEN SOURCE PUBLIC MODEL ARCHIVE

Savage
Scenario Authoring and Visualization for Advanced Graphical Environments

The Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE) project is a large archive of dynamic 3D military models and authoring tools, all open source and built using Extensible 3D (X3D) graphics.

Zip archive 33 Sections, 160 Chapters, 1079 Models Help

Aircraft Fixed Wing | Aircraft Helicopters | Aircraft Miscellaneous | Amphibious Vehicles | Auv Workbench | Avatars | Biologics | Buildings | Communications And Sensors | Environment | Ground Vehicles | Harbor Equipment | Harbors | Locations | Model Detailing | Offshore Structures | Robots | Scenarios | Ships Civilian | Ships Military | Space | Submarines | Tools | Weapons

Aircraft Fixed Wing

AV 8 B - Harrier - United States	Bear - Russia	C 130 - Hercules - Tunisia
Catalina	Euro Fighter	F 16 - Fighting Falcon - Turkey
F 18 - Blue Angel - United States	F 18 - Superhornet - United States	Jhl Heavy Lift - NPS
Mv 22 - Osprey - United States	P 3 Orion	

Aircraft Helicopters

AH 1 Super Cobra - United States	AH 1 W - United States	AH 64 DApache Longbow - United States
CH 46 E - Sea Knight - United States	CH 53 - United States	Helicopter - United States
Helix - Russia	Jhl Heavy Lift - NPS	MH 53 ESea Dragon - United States
OH 58 D - Kiowa Warrior - United States	SH 60 - Seahawk - United States	SH 60 B
SH 60 B - Seahawk -		

United States

Aircraft Miscellaneous

Balloon	Blimp	Zeppelin

Amphibious Vehicles

AAAV	AAV	LCAC

Auv Workbench

AVCL

Avatars

Marines

Biologics

Dolphin

Buildings

Erdc Two Story Building	House Baris Aktop	House Seksit Siripala
Playground	Soccer Stadium	UHRB
Zen Condominium		

Communications And Sensors

Beam	Half Dome	Omni Directional
Satellite	Sea Web	Sonar
Sonobuoys	TRC 170	TSSR
WISP		

Environment

Oceanography	Sea State	Time Of Day

Ground Vehicles

BMP 1	HMMWV	Jeep
M 1 A 1	M 1 A 2	M 2 A 3
M 577	MEFFV	MLRS 270
T 72 M	Wolverine	

Harbor Equipment

Barrier	Brow	Buoys
Canopy	Chain Link Fence	Pier Riser
Table	Waterline Security Light	

Harbors

Equipment

Locations

Camp Pendleton California	Fort Lauderdale Florida	Hawaii
Naval Postgraduate School	Port Hueneme California	Rio De Janeiro
San Francisco California	Ship Island Mississippi	Southern California Border

Model Detailing

Hull Numbers

Offshore Structures

Oil Rigs

Robots

Jet Fire Transformer Toy	Unmanned Air Vehicles	Unmanned Ground Vehicles
Unmanned Surface Vehicles	Unmanned Underwater Vehicles	

Scenarios

Amphibious Raid Camp Pendleton	Capture The Flag	Collision Uss Greeneville Mv Ehime Maru
Jfcom Dcee Exercise July 2003	Limited Objective Experiment Port Hueneme	Remus Mission 10 MAR 2003
Tank Maneuver	Uss Cole Terrorist Attack	UW 3303 Minefield Search

Ships Civilian

Barge	Cargo Ships	Cigarette Boat
Cruise Ship	Ferries	Hovercraft - Singapore -

		SNR 6
Merchant Livestock Carrier	Personal Water Craft	Sail Boats
Small Craft	Supertanker	Trawlers
Tugboats		

Ships Military		
Carrier - Independence - United States	Carrier - Nimitz - United States	Carrier - Saratoga - United States
Cruiser - United States	DD 963 - Spruance - United States	DDG - 51 Flight IIA - United States
DDG - Arleigh Burke - United States	Destroyer - Sovremenny - Russia	FFG - 7 Oliver Hazard Perry - United States
FFG - Oilver Perry - United States	FFG - Oliver Perry - United States	Frigate - Greece
Frigate - Greece - MEKO 200	Frigate - Yavuz - Turkey	Harbor Ferry Boat
Hovercraft - Singapore - SNR 6	Landing Platform Dock - LPD	Landing Ship Tank - Endurance - Singapore
Large Deck Amphib - Boxer - United States	Missile Attack Boat Osa II	Patrol Craft - Greece - Vosper
Patrol Craft - Russia - Nanuchka - Lighthouse	Patrol Craft - United States - Tempest	RHIB - United States
Running Lights	Sea Base	

Space		
Shuttle	Solar System	Space Attack

Submarines		
Periscope Reticle	SSGN - Ohio - United States	SSN - Los Angeles - United States
Various		

Tools		
Animation	Authoring	Exercise Clock
Explosions	Heads Up Displays	SMAL
Symbology	Terrain	VRML 1

Weapons		
Ammunition	Crew Served Weapons	Guns

Missiles Small Arms Torpedoes
Underwater Mines

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. COMPLETED MASTER'S THESIS RESEARCH TOPICS USING SIMKIT

- Mounir , MAJ Tunis Air Force, A Teamwork Air Traffic Control (ATC) Simulator (accessed September 2006)
- Christopher J. Nannini, CAPT U.S. Army, Assignment Scheduling Capability for UAVs (ASC-U) Simulation Tool (accessed September 2006)
- Lehmann, Wolfgang, MAJ German Army An Upgradeable Agent-Based Model To Explore Non-Linearity And Intangibles In Peacekeeping Operations (accessed September 2006)
- ALLEN, TIM, LT U.S. Navy, "Using Discrete Event Simulation To Assess Obstacle Location Accuracy In The REMUS Unmanned Underwater Vehicle" (accessed September 2006)
- REVOR, MARK, Capt., USMC, "An Analysis Of The Integrated Mechanical Diagnostics Health And Usage Management System On Rotor Track And Balance"(accessed September 2006)
- SCHOCH, ERIC, LCDR U.S. Navy, "A Simulation Of The I3 To D Repair Process And Sparing Of The F414-Ge-400 Jet Aircraft Engine" (accessed September 2006)
- MARGOLIS, MICHAEL, Captain, U.S. Marine Corps, "Operational Availability and Cost Trade-Off Analysis for the Multi-Mission Maritime Aircraft" (accessed September 2006)
- NAWARA, TERRENCE, LT U.S. Navy, "Tactical Route Planning for Submarine Mine Detection and Avoidance." (accessed September 2006)
- DICKIE, ALISTAIR, Captain, Australian Army, "Modeling Robot Swarms Using Agent-Based Simulation" (accessed September 2006)
- HAVENS, MICHAEL E., Lieutenant, U.S. Navy, "Dynamic Allocation of Fires and Sensors," (accessed September 2006)
- CHILDS, MATTHEW D., Lieutenant Commander, U.S. Navy, "An Exploratory Analysis of Water Front Force Protection Measures Using Simulation," (accessed September 2006)

- ERLENBRUCH, THOMAS, Captain, German Army, “Agent-based Simulation of German Peacekeeping Operations for Units up to Platoon Level,” (accessed September 2006)
- FRICKE, CAROLYN S., Lieutenant Commander, U.S. Navy, “Operational Logistics Wargame,” (accessed September 2006)
- SAN JOSE, ANGEL E., Lieutenant Commander, Spanish Navy, “Analysis, Design, Implementation and Evaluation of Graphical Design Tool to Develop Discrete Event Simulation Models Using Event Graphs and Simkit,” (accessed September 2006)
- LENHARDT, THOMAS A., Captain, U.S. Marine Corps, “Evaluation of Combat Service Support Logistics Concepts for Supplying a USMC Regimental Task Force,” (accessed September 2006)
- MAJ Ronald F. Woodaman, United States Marine Corps, "Agent-Based Simulation of Military Operations Other Than War Small Unit Combat" (accessed September 2006)
- MAJ Thomas E. Turner, United States Marine Corps, "A Simulation of the Joint Tactical Radio System Bandwidth Requirements to Support Marine Coprs Ship-To-Objective Maneuver in 2015" (accessed September 2006)
- LT Patrick V. Mack, United States Navy, "THORN: A Study In Designing A Usable Interface For A Geo-Referenced Discrete Event Simulation" (accessed September 2006)
- CDR Knut Armo, Norwegian Navy, "The Relationship Between A Submarine’s Maximum Speed And Its Evasive Capability" (accessed September 2006)
- LT Hyung Le, United States Navy, "Advanced Naval Surface Fire Support Weapon Employment Against Mobile Targets" (accessed September 2006)
- LTJG Erhan Aidin, Turkish Navy, "Screen Dispositions of Naval Task Forces against Anti-Missile Ships"
- MAJ David P. Krizov, United States Marine Corps, "Tactical Exercise Review and Evaluation System"
- LT John R. Sterba ,United States Navy, "Operational Maneuver from the Sea Logistics Training Aid"

- LT Anthony W. Troxell, Lieutenant, United States Navy, "Naval Logistics Simulator"
- CDR Inge A. Utaaker, Norwegian Navy, "Distribution of Firing Directions in a Coordinated Surface-to-Surface Missile Engagement,"
- CAPT Mark A. Grabski, United States Army, "Assessing the Effectiveness of the Battlefield Combat Identification System" (accessed September 2006)
- CAPT Garrett D. Heath, United States Army, "Simulation Analysis of Unmanned Aerial Vehicles (UAV)" (accessed September 2006)
- CAPT Dale L. Henderson United States Army, "Modterrain: A Proposed Standard for Terrain Representation in Entity Level Simulation" (accessed September 2006)
- MAJ William Bohman, USA, "STAFFSIM, An Interactive Simulation for Rapid, Real Time Course Of Action Analysis By U.S. Army Brigade Staffs" (accessed September 2006)
- CAPT Michael W. Rauhut, USA, "Automating A Study Question Methodology To Enhance Analysis In High Level Architecture" (accessed September 2006)
- CAPT Keith A. Hattes, USA, "Special Operations Mission Planning and Analysis Support System" (accessed September 2006)
- CAPT Norbert Schrepf, German Army, "Visual Planning Aid For Movement Of Ground Forces In Operations Other Than War" (accessed September 2006)
- LT Phillip Pournelle, USN, "Component Based Simulation Of The Space Operations Vehicle And The Common Aero Vehicle" (accessed September 2006)
- LTCDR James Townsend, USN, "Defense of Naval Task Forces from Anit-Ship Missile Attack" (accessed September 2006)
- MAJ Arent Arntzen, RNAF, "Software Components For Air Defense Planning" (accessed September 2006)
- LCDR John Ruck, USN, "An Object-Oriented Discrete-Event Simulation of Logistics" (accessed September 2006)
- CAPT Steven D. Knight, USA, "A Comparison of Analysis in DIS and HLA" (accessed September 2006)

- LCDR Arthur Cotton, USN, "Standard Theater-Level Army Combat Modeling and Simulation Objects" (accessed September 2006)
- CAPT Douglas Dudgeon, USMC, "Standard Army Combat Modeling and Simulation Objects" (accessed September 2006)
- LT Steven Kinskie, USN, "An Evaluation of the Budget and Readiness Impacts of Battlegroup Sparing" (accessed September 2006)
- LT Andrew Stewart, USN, "Evaluating the Benefits of a TDOA/FDOA GPS-Assisted Geolocation System" (accessed September 2006)
- MAJ Paul Warhola USMC, "Assessment of a Hybrid Ground-Ground/Air-Ground Attrition Adjudication Methodology" (accessed September 2006)
- LT Max Willey, USN, "A Sea-Based Combat Logistics Concept for Marine Expeditionary Forces" (accessed September 2006)
- CAPT Larry Larimer, USA, "Building an Object Model of a Legacy Simulation" (accessed September 2006)
- LT Joe Huffaker, USN, "SupportingManuever Warfare from Forward Logistics Bases" (accessed September 2006)
- LT Kirk Stork, USN, "Sensors In Object Oriented Discrete Event Simulation" (accessed September 2006)
- LT Mike Walls, USN, "Improving the Aviation Depot Induction Process Through Planned Depot Maintenance" (accessed September 2006)

APPENDIX F. DISTRIBUTION AND SOURCE CODE ACCESS

This appendix provides amplifying information on obtaining the source code, including examples, of the work that was done in this thesis.

The Simkit API is available at http://diana.cs.nps.navy.mil/Simkit/ .

Viskit is available via password protected access at https://diana.cs.nps.navy.mil/Viskit/Viskit-0.2.8/install.htm (accessed August 2006)

The majority of 3D models used in this work are publicly available as part of the SAVAGE modeling archive located at: https://savage.nps.edu/Savage (accessed August 2006). Restricted password protected models used in this thesis are accessible at https://savagedefense.navy.mil/SavageDefense/ (accessed August 2006). Access to this FOUO repository can be requested by contacting:

Dr. Don Brutzman: Brutzman@nps.edu;

Research Associate Curtis Blais: clblais@nps.edu

Research Associate Terry Norbraten: tdnorbra@nps.edu .

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Akpan, Justice I., and Roger J. Brooks. 2005. *Practitioners perception of the impacts of virtual reality on discrete-event simulation*. *Proceedings of the 2005 Winter Simulation Conference* (December).

Ames, Andrea L., David R. Nadeau, and John L. Moreland. 1997. *VRML 2.0 sourcebook*. Second ed. New York, NY: John Wiley & Sons.

Brutzman, D. *X3D scene authoring hints*. 2006 [cited September 7, 2006]. Available from http://www.web3d.org/x3d/content/examples/X3dSceneAuthoringHints.html.

———. 2003. *Web-based 3D graphics rendering of dynamic deformation structures in large-scale distributed simulations.* Monterey, California: Naval Postgraduate School, https://www.movesinstitute.org/xmsf/projects/X3D/DeformableSurfacesTechnicalReport2003November.pdf (accessed September 2006).

———. 2002. *X3D-Edit authoring tool for extensible 3D (X3D) Graphics.* Monterey, California: Naval Postgraduate School, http://www.web3d.org/x3d/content/X3D-EditAuthoringTool.pdf (accessed September 2006).

Brutzman, D. P., C. L. Blais, and T. D. Norbraten. 2006. *Naval installation security modeling and simulation workshop summary report.* Monterey, California..

Brutzman, D. P., L. Daly. 2006. *Extensible 3D (X3D) graphics for web authors*. Morgan-Kauffman Publishers.

Brutzman, D., K. Morse, M. Pullen, and M. Zyda. 2002. *XMSF findings and recommendations report.* Monterey, California. https://www.movesinstitute.org/xmsf/XmsfWorkshopSymposiumReportOctober2002.pdf (accessed August 2006).

Buss, Arnold H. 2004. *Simkit analysis workbench for rapid construction of modeling and simulation components*. *Proceedings of the Fall Simulation Interoperability Workshop* (September), http://diana.gl.nps.navy.mil/~ahbuss/papers/Discrete%20Event%20Programming%20with%20Simkit.pdf (accessed August 2006).

———. 2001. *Discrete event modeling with Simkit*. *Simulation News Europe* (November), http://diana.gl.nps.navy.mil/~ahbuss/papers/BasicEventGraphModeling.pdf (accessed August 2006).

Buss, Arnold H., and Roberto Szechtman. 2006. *OA3302 system simulation, course notes*, http://diana.gl.nps.navy.mil/oa3302/ (accessed August 2006).

Chief of Naval Installations. 2005. *Navy ashore vision 2030: Naval installations - the foundation of readiness.*, http://www.cnic.navy.mil/CNIC_HQ_Site/AboutCNIC/GeneralInformation/RelatedDocuments/NAV_2030_FINAL (accessed August 2006).

Cioppa, Thomas M. 2002. *Efficient nearly orthogonal and space-filling experimental designs for high-dimensional complex models*. Dissertation., Naval Postgraduate School, http://library.nps.navy.mil/uhtbin/hyperion-image/02sep%5FCioppa%5FPhD.pdf (accessed August 2006).

Congressional Reporting Service (CRS). 2001. *Terrorist attack on USS Cole: Background and issues for congress.* Washington, D.C.:, http://news.findlaw.com/cnn/docs/crs/colterrattck13001.pdf (accessed August 2006).

Darken, Chris. 2005. *MV4025 cognitive and behavioral modeling for simulations, course notes*.

Ferber, Jacques. 1999. *Multi-agent system, an introduction to distributed artificial intelligence*. Harlow: Addison-Wesley Publishers.

Harney, James W. 2003. *Analyzing anti-terrorist tactical effectiveness of picket boats for force protection of navy ships using X3D graphics and agent-based simulation*. Master's Thesis., Naval Postgraduate School, http://library.nps.navy.mil/uhtbin/hyperion/03Mar%5FHarney.pdf (accessed August 2006).

Hiles, John. 2006. *MV4015, Agent-based autonomous behavior for simulations course notes*.

Hunsberger, M. G. 2001. *3D visualization of tactical communications for planning and operations using virtual reality modeling language (VRML) and extensible 3D (X3D)*. Master's Thesis., Naval Postgraduate School.

Murray, Mark W., and Jason M. Quigley. 2003. *Automatically generating a distributed 3D battlespace using USMTF and XML-MTF air tasking order, extensible markup language (XML) and virtual reality modling language (VRML)*. Master's Thesis., Naval Postgraduate School, http://library.nps.navy.mil/uhtbin/hyperion/00Jun_MurrayM.pdf (accessed August 2006).

Nicklaus, Shane D. 2003. *Scenario authoring and visualization for advanced graphical environments*. Master's Thesis., Naval Postgraduate School, http://library.nps.navy.mil/uhtbin/hyperion/01Sep_Nicklaus.pdf (accessed August 2006).

Osborn, Brian. 2002. *An agent-based architecture for generating interactive stories*. Dissertation., Naval Postgraduate School, http://library.nps.navy.mil/uhtbin/hyperion-image/02sep%5FOsborn%5FPhD.pdf (accessed August 2006).

Rauch, Travis M. 2006. *Savage modeling and analysis language (SMAL) : Metadata for tactical simulations and X3D visualizations*. Master's Thesis., Naval Postgraduate School.

Schruben, L. 1983. *Simulation modeling with event graphs*. *Communications of the ACM* 26, : 957.

Shafika, Mattar. 2005. *U.S. ship attacked in jordan port rockets miss mark; no sailors injured*. *Washington Post*, August 20, 2005.

Singhal, Sandeep, and Michael Zyda. 1999. *Networked virtual environments: Design and implementation* Addison-Wesley Publishers.

White House. 2005. *National strategy for maritime security.* , http://www.whitehouse.gov/homeland/maritime-security.html.

Wu, Thomas C. 2004. *An introduction to object-oriented programming with java*. Third Edition. New York, New York: McGraw-Hill.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Don Brutzman
   Naval Postgraduate School
   Monterey, California

4. Curt Blais
   Naval Postgraduate School
   Monterey, California

5. John Moore
   Navy Modeling and Simulation Office
   Ft. Belvoir, Virginia

6. John Hiles
   Naval Postgraduate School
   Monterey, California

7. Arnold Buss
   Naval Postgraduate School
   Monterey, California

8. Joseph McConnell
   Naval Facilities Engineering Command
   Washington Navy Yard, District of Columbia

9. CAPT James D. Scola, USN
   Commander, United States Pacific Fleet
   Aiea, Hawaii

10. Caroline Massie
    Chief of Naval Installations
    Anacostia Annex, District of Columbia

11. Dr. Brian Donahue
    Johns Hopkins University
    Baltimore, Maryland

12. Dr. James D. Miller
Johns Hopkins University
Baltimore, Maryland

13. CAPT Taylor Skardon
Naval Station Pearl Harbor
Pearl Harbor, Hawaii

14. Alexandra De Visser
Naval Facilities Engineering Service Center
Port Hueneme California

15. Dallas Meggitt
Sound and Sea Technology
Edmunds, Washington

16. Dennis Garrood
Sound and Sea Technology
Edmunds, Washington

17. Mario Pozzo
Sound and Sea Technology
Edmunds, Washington

18. Gene Hackney
Naval Air Systems Command
Patuxent River, Maryland

19. Len Daly
Daly Realism
Valley Glen, California

20. Rick Goldberg
Aniviza Inc.
Los Gatos, California

21. LT Wilfredo Cruzbaez
Naval Postgraduate School
Monterey, California

22. Alan Hudson
Yumetech, Inc.
Seattle, Washington

23. Terry Norbraten
Naval Postgraduate School
Monterey, California

24. CDR John Kennington
Commander, United States Pacific Fleet
Aiea, Hawaii

25. Wendy Walsh
Naval Postgraduate School
Monterey, California

26. John Seguin
Naval Postgraduate School
Monterey, California

27. Mark W. Kenny
Center for Submarine Counter-Terrorism Operations
Groton, Connecticut

28. CAPT Rick J. Ruehlin
Littoral and Mine Warfare
Washington Navy Yard, District of Columbia

29. F. P. Gustavson
Leadership and Management Solutions
Kailua, Hawaii

30. Richard L. Snead
National Security Directorate
Oak Ridge, Tennessee

31. Kelvin Ogata
State of Hawaii, Department of Transportation
Honolulu, Hawaii